VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Multiplatformní vývoj v prostředí Xamarin

Multiplatform Development with Xamarin

2018 Bc. Jiří Hopják

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student:	**Bc. Jiří Hopják**
Studijní program:	N2647 Informační a komunikační technologie
Studijní obor:	2612T025 Informatika a výpočetní technika
Téma:	Multiplatformní vývoj v prostředí Xamarin Multiplatform Development with Xamarin
Jazyk vypracování:	čeština

Zásady pro vypracování:

Cílem práce je zhodnotit současné možnosti vývoje aplikací pro různé platformy, a to s využitím vývojového prostředí Xamarin. Práce poskytne přehled o současných možnostech a technologiích a současné také reálnou implementaci aplikace.

1. Zmapujte oblast multiplatformních prostředků pro vývoj aplikací.
2. Detailně popište koncepty vývoje aplikací v prostředí Xamarin.
3. Navrhněte aplikaci, která bude ilustrovat vývoj na platformě Xamarin a bude následně reálně využitelná, např. vizualizace informací s využitím API.
5. Implementujte aplikaci s důrazem na její následné multiplatformní nasazení.
6. Zhodnoťte prostředí Xamarin z hlediska vývoje i nasazení aplikace.

Seznam doporučené odborné literatury:

[1] Jon Duckett: JavaScript and JQuery: Interactive Front-End Web Development, Wiley, 2014, ISBN: 978-1118531648
[2] Sasha Vodnik: HTML5 and CSS3, Illustrated Complete, Course Technology, 2015, ISBN: 978-1305394049
[3] Jason Beaird: The Principles of Beautiful Web Design, SitePoint, 2014, ISBN: 978-0992279448
[4] Erixc Elliot: Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries, O'Reilly Media, 2014, ISBN: 978-1491950296

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Michal Radecký, Ph.D.**

Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 14. dubna 2018 .....

Rád bych na tomto místě poděkoval Ing. Michalu Radeckému, Ph.D. za odbornou pomoc a konzultaci při zpracovávání této diplomové práce.

Abstrakt

Tato diplomová práce pojednává o současných technologiích pro tvorbu multiplatformních aplikací se zaměřením na framework Xamarin. Mezi hlavní cíle této práce patří především představení a specifikování multiplatformních možností z pohledu vývoje. Jako ilustrační nástroj vývoje následně poslouží navržená multiplatformní aplikace využívající webové API rozhraní pro zobrazování dat z rezervačních systémů.

Klíčová slova: Xamarin, Xamarin.Forms, Multiplatformní vývoj, Android, iOS, MVVM,.NET, C#, XAML, API, JSON, Visual Studio

Abstract

This thesis deals with the current technologies for development of multiplatform applications focusing on Xamarin framework. The main objectives of this work includes the introduction and specification of multiplatform possibilities from development point of view. As an illustrative development tool will be created multiplatform application using the Web API interface for displaying data from reservation systems.

Key Words: Xamarin, Xamarin.Forms, Multiplatform development, Android, iOS, MVVM,.NET, C#, XAML, API, JSON, Visual Studio

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CSS	–	Cascading Style Sheets
HTML	–	HyperText Markup Language
JSON	–	JavaScript Object Notation
MSDN	–	Microsoft Developer Network
XAML	–	Extensible Application Markup Language
XML	–	Extensible Markup Language
HTTP	–	Hzpertext Transfer Protocol
SDK	–	Software Development Kit
OS	–	Operating System
DTO	–	Data Transfer Object
UWP	–	Universal Windows Platform
CPU	–	Central Processing Unit
IDE	–	Integrated Development Environment
ARM	–	Advanced RISC Machine
PCL	–	Portable CLass Library
WPF	–	Windows Presentation Foundation
UML	–	Unified Modeling Language
LCD	–	Liquid Crystal Display
MVVM	–	Model-View-ViewModel

Seznam obrázků

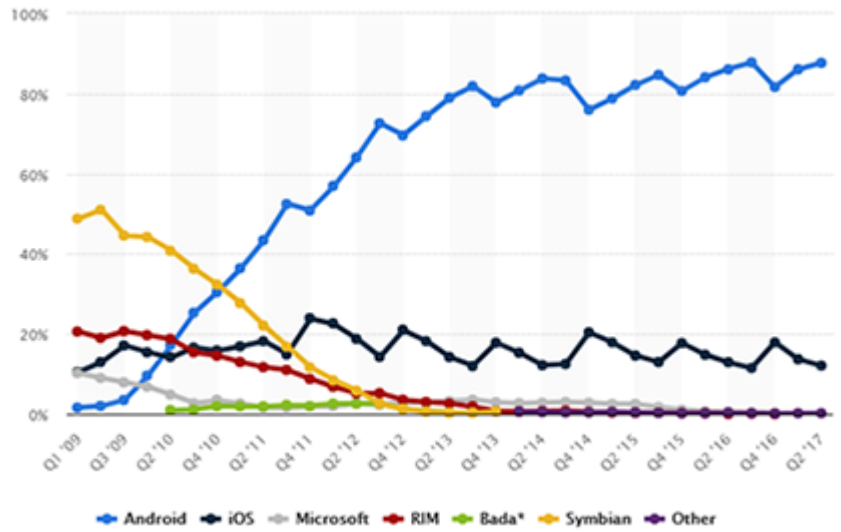
Seznam výpisů zdrojového kódu

1 Úvod

V posledních několika letech si mnozí nedokážeme představit život bez moderní techniky. Nejrůznější elektronická zařízení se stala součástí našeho běžného života, především přístroje jako jsou chytré mobilní telefony, tablety a jiná podobná zařízení. Mezi charakteristické prvky těchto přístrojů můžeme označit především velký dotykový displej a chytrý operační systém. Trendem poslední doby je také neustálé zvyšování výkonu zařízení, ať už se jedná o rychlost procesoru, nebo velikost interní a RAM paměti. V současnosti můžeme na výkonnosti vidět obrovské pokroky, které mnohdy i mnohonásobně překonávají kancelářské notebooky nebo počítače.

Nedílnou součástí chytrých zařízení jsou uživatelské aplikace a právě v této oblasti se naskytuje potencionálně velká šance pro vývojáře srovnat nabídku s poptávkou. Bohužel zde vzniká nevýhoda, protože uživatelé těchto zařízení jsou rozděleni do několika platforem. Pro vývojáře to znamená, že jestliže chtějí uspokojit všechny zákazníky a nechtějí nějaké uživatele ochudit, musejí být schopni dodat svůj produkt na všechny platformy nebo alespoň na několik nejvíce používaných z pohledu podílu na trhu. Protože každá platforma využívá jiné technologie pro implementaci aplikací, je v současné době několik možností, jakým způsobem můžeme tyto aplikace vyvíjet. Jednou z možností je separátní vývoj pro jednotlivé platformy zvlášť. Tento způsob nám však značně navyšuje náklady spojené s vývojem. Druhým způsobem a mnohdy do značné míry i výrazně levnějším, je využití multiplatformního vývoje, který je založen na univerzálním kódu a může vést ke zjednodušení a zrychlení samotného vývoje aplikace.

Na začátku roku 2018 je podíl na celosvětovém trhu s chytrými operačními systémy v mobilních zařízeních zastoupen třemi platformami. Mezi ty nejrozšířenější patří zejména systémy Android od společnosti Google, iOS systém od Applu a z menší části ještě Windows 10 Mobile vyvíjený společností Microsoft.[1]



Obrázek 1: Zastoupení mobilních operačních systémů v posledních několika letech. [1]

V této práci se budu zabývat možnostmi multiplatformního vývoje aplikací pro mobilní systémy a především se zde budu podrobněji věnovat multiplatformnímu vývoji v prostředí Xamarin. Zároveň po prostudování tohoto prostředí ukáži reálnou implementaci aplikace, která ilustruje vývoj ve zmíněném prostředí Xamarin.

Práce je rozložena do několika kapitol, nejdříve je popsán základní přehled o třech nejrozšířenějších mobilních platformách, možnosti vývoje jejich aplikací a následná publikace pro sdílení aplikace směrem k systémovým uživatelům. Součástí kapitoly je také popis možností multiplatformního vývoje, na který navazuje kapitola věnující se vývoji v technologii Xamarin. Po teoretické části práce následuje kapitola zabývající se prvotní analýzou a popisem specifikací pro implementovanou aplikaci. V páté kapitole již následuje samotná implementace aplikace, kde jsou ukázky kódu popisující práci s daty, návrhovými a architektonickými vzory. Tato kapitola je rozdělena do sekcí popisující multiplatformní vývoj samotné aplikace a současně také nativní funkcionality každé platformy. Následující dvě kapitoly se zabývají detailním popisem procesu publikace pro systém Android a iOS, který je rozložen do několika podkapitol. Vzhledu a ukázce obrazovek výsledné aplikace je věnována samostatná kapitola, která představuje funkčnost aplikace na platformě Android. Na závěr této práce je následné zhodnocení celého prostředí, které bylo využito při implementování multiplatformní aplikace.

1.1 Cíl práce

Cílem této práce je přiblížit a zhodnotit současné možnosti vývoje aplikací pro různé druhy platforem. Především však detailně prozkoumat a popsat možnosti využití vývojového prostředí Xamarin. Práce následně poskytne přehled o současných možnostech a vývojových technologiích, ale především bude dbát na reálnou implementaci aplikace (např. vizualizace informací s využitím API rozhraní), která bude následně reálně využitelná. Tato aplikace bude ilustrovat vývoj v multiplatformním prostředí Xamarin, který bude detailně popsán a zobrazen na ukázkách kódů.

2 Mobilní platformy

Jako mobilní platformu si můžeme představit především tzv. chytré telefony, phablety, tablety nebo také jiné zařízení, jako jsou např. chytré brýle, hodinky, televize s operačním systémem a nově také i systémy v autech. Jak už bylo řečeno v úvodní kapitole, mezi nejrozšířenější mobilní platformy patří systémy Android, iOS a Windows 10 Mobile, přičemž poslední z uvedených má pouze malé procentuální zastoupení na trhu a proto lze říct, že výraznou většinu ovládají systém Android a iOS. Procentní zastoupení těchto dvou systému se může výrazně lišit a to především z pohledu různých zemí, kdy například ve Spojených státech amerických má vyšší podíl na trhu operační systém iOS, ale naopak v ostatních zemích světa převažují zařízení s Androidem.

2.1 Android

Operační systém Android je založen na jádru operačního systému Linux, který je dostupný a šířen ve formě otevřeného zdrojového kódu (open source). Jeho vývoj má na starosti firma Google, která se pod hlavičkou konsorcia firem Open Hanset Alliance, stará především o vývoj a údržbu jádra tohoto systému. Samotní výrobci zařízení mohou následně využívat systém Android a upravovat pro svoji potřebu při dodržení předem stanovených podmínek. Například při větších změnách systému je nutné změnit také jeho název.[2]

Jak bylo dříve zmíněno, systém Android má dlouhodobě mezi všemi operačními systémy největší zastoupení na světě. V současnosti, na začátku roku 2018 je nejnovější verze systému označena jako Android 8.0 Oreo.[3]

2.1.1 Vývoj aplikací pro Android

Jeden z hlavních důvodů oblíbenosti mezi vývojáři aplikací pro systém Android, je možnost vyvíjet v programovacím jazyku Java, který je jedním z nejrozšířenějších programovacích jazyků na světě. Nutno však dodat, že vývoj pro platformu Android se s vývojem standardních Java aplikací mírně liší, ale pouze o malé drobnosti, jako je např. životní cyklus aplikace.

Mezi oficiálně podporované vývojové prostředí patří Eclipse, který s doinstalovaným ADT plugin je velmi kvalitní pomocník při vývoji a ulehčuje práci s Android projektem. Druhým vývojovým prostředím, které je taktéž velmi kvalitní pro implementaci aplikací, je Android studio spravováno společností Google. V poslední době se však začíná toto vývojové prostředí dostávat do popředí a odsunuje Eclipse studio na druhé místo.

Jestliže vývojář nechce používat pro vývoj ani jedno z uvedených prostředí, má možnost využít jiného IDE nebo např. použít jednoduchý textový editor a pro následnou kompilaci aplikace využít pomoc příkazového řádku.[4]

2.1.2 Distribuce Android aplikací

Distribuce aplikací pro systém Android k jednotlivým zařízením je možné provést primárně přes oficiální web Google Play. Jedná se o internetový katalog pro stahování aplikací a her, který provozuje přímo společnost Google. Aplikace do katalogu je možné vkládat, skrze vytvořený účet s vývojářskou licencí. Tuto licenci může získat každý po zaplacení jednorázové částky 25 USD. Po přidání do katalogu projde aplikace schvalovacím procesem, který se provádí automatizovaně a následně je aplikace dostupná do několika hodin od první publikace.

Kromě oficiálního katalogu Google Play, existují spousty neoficiálních repositářů, kde je možné také stáhnout a nainstalovat do zařízení tzv. svobodné aplikace. Mezi nejznámější repositáře aplikací patří např. Amazon AppStore, F-Droid nebo AndroidMarket.

Protože systém Android je otevřená platforma, existuje zde také možnost nahrát a nainstalovat aplikace přímo ze zařízení nebo počítače. Pro tuto možnost je však nutné souhlasit a potvrdit odemknutí systémového módu pro vývojáře, který lze aktivovat několika způsoby přímo v nastavení operačního systému.[5]

Obrázek 2: Android logo [6]

2.2 iOS

Operační systém iOS zvaný v dřívějších verzích jako iPhone OS. Jedná se o druhý nejrozšířenější mobilní operační systém, který byl vytvořen a je stále vyvíjen společností Apple. Tento systém byl v jeho začátcích určen původně pouze pro mobilní telefony iPhone, ale v pozdějších letech se začal používat také i na dalších zařízení této firmy, jako jsou např. tablety iPad, přístroje pro přehrávání hudby iPod Touch a nejnověji také v Apple TV.

Na rozdíl od Androidu volí společnost Apple úplně odlišnou politiku. Samotný systém totiž není otevřený zdrojový kód, ale naopak je uzavřený, bez možnosti k náhledu na zdrojový kód. Menší překážkou pro vývojáře je také větší omezení systému a to především možnost přístupu aplikací k systémovým rozhraním a využití zdrojů.[7]

2.2.1 Vývoj aplikací pro iOS

U iOS systému dlouho dobu převládal vývoj pomocí jazyka C nebo pokročilejšího Objective-C. Současně bylo také v minulých letech možné vyvíjet pouze v prostředí XCode. Toto prostředí je však dostupné pouze z operačních systémů Mac OS X a proto nebylo možné vyvíjet z operačních systémů Windows nebo Linux. V posledních letech však společnost Apple umožnila vývojářům vyvíjet i v jiném operačním systému než je Max OS X a také představila nový moderní programovací jazyk Swift, který je určen pro vývoj aplikací systému iOS. Tento jazyk je publikován jako open source a jeho zdrojové kódy jsou volně k dispozici. Tento krok vyvolal u vývojářů velké nadšení, protože umožňuje samotným vývojářům návrh k vylepšení jazyka do dalších verzí.[8]

2.2.2 Distribuce iOS aplikací

AppStore je katalog pro distribuci iOS aplikací. Na rozdíl od Androidu je to pouze jediný katalog pro distribuci aplikací a neexistuje zde jiná legální možnost jak nainstalovat aplikaci z jiného zdroje než je AppStore. Jestliže chceme distribuovat aplikaci, je potřeba počítat s daleko přísnějšími podmínkami pro publikaci než u Android systému. Jelikož samotnou aplikaci schvaluje reálná osoba, je nutné počítat se schvalovacím procesem, který může trvat i několik týdnů. Tento způsob distribuce aplikací je sice do značné míry omezující pro vývojáře, ale jestliže se na to podíváme z druhé strany, tento proces vede k výrazně lepší bezpečnosti a stabilitě celého systému. Na rozdíl od Androidu se také výrazně liší cena vývojářského účtu, kdy je potřeba zaplatit vývojářský poplatek 99 USD a platí se každý rok.[9]

Obrázek 3: iOS logo [6]

2.3 Windows 10 Mobile

Mobilní operační systém Windows 10 Mobile od Microsoftu je nástupce méně úspěšné řady Windows Phone (v poslední verzi 8.1). Tento operační systém byl vyvinut s cílem sjednotit platformu aplikací a tím přinést možnost zobrazovat aplikace na více různých zařízeních. Hlavní charakteristika je poskytování rozsáhlejší synchronizace dat napříč jednotlivými zařízeními s Windows 10, jakou jsou např. stolní počítače, mobilní zařízení nebo konzole Xbox.[10]

Systém Windows 10 Mobile je určen především pro použití na chytrých mobilních telefonech, phabletech a tabletech s architekturou procesoru ARM. První zařízení s tímto operačním systémem byly chytré telefony značky Lumia, které se začaly vyrábět koncem roku 2015. Některé starší verze těchto mobilních telefonů měly aktualizaci na nový operační systém zdarma.

Na podzim roku 2017 bylo vydáno prohlášení, že společnost Microsoft přerušila vývoj systému Windows 10 Mobile z důvodu nízkého podílů na trhu a nedostatečného vývoji aplikací ze strany vývojářů. Pro udržování platformy (do doby než přijde nový nástupce), bude společnost Microsoft vydávat pouze opravy balíčků a základní údržbu této platformy.[11]

2.3.1 Vývoj aplikací pro Windows 10 Mobile

Pro vývoj univerzálních aplikací pro Windows 10 je nutné, aby vývojář měl nainstalovaný operační systém Windows 10 a také vývojové prostředí Visual Studio 2015 nebo vyšší. Velkou výhodou při vývoji je také podpora procesoru s technologií Hyper-V pro spuštění a testování aplikací na emulátoru operačního systému, který slouží jako plnohodnotný zástupce fyzického zařízení. Jestliže procesor nesplňuje tuto podmínku, je nutné veškeré testování a kalibrace provádět na fyzickém zařízení.[12]

Na rozdíl od systému Android nebo iOS je vývoj univerzálních aplikací pro Windows 10 značně volnější, alespoň co se týče výběru programovacího jazyka. Vývojáři si mohou vybrat z několika víceúrovňových jazyků jakou jsou C++, C#, Visual Basic nebo Javascript. Javascript je jediným jazykem využívající HTML5 pro vytváření a rozkládání uživatelského rozhraní, ostatní jazyky používají značkovací jazyk zvaný XAML. Každý z těchto jazyků je vhodnější pro různé typy aplikací, např. jestliže je důležitý výkon aplikace, je vhodné použít jazyk C++. Naopak pro univerzální využití aplikace je nejpopulárnější mezi vývojáři jazyk C#.[13]

2.3.2 Distribuce Windows 10 Mobile aplikací

Publikace univerzálních aplikací probíhá prostřednictvím služby Microsoft Store, původně nazývaný Windows Store. Jedná se o obchod s digitální obsahem pro aplikace a hry. V zahraničí je zde možné najít i multimediální obsah jako jsou filmy, hudba nebo seriály. Vzhledem k výše zmiňovaných platforem, je cena pro vystavení aplikace pouze 19 USD a platí se pouze jednorázově pomocí platební karty.

Proces publikace aplikace může probíhat v obchodu několika způsoby. Mezi základní možnosti patří např. distribuce aplikace pouze pro vybranou skupinu uživatelů nebo díky offline licencím se nabízí nově možnost instalace aplikací bez přístupu k Microsoft Store. Podobně také jako u Apple Storu je schvalovací proces řízený reálnou osobou, která kontroluje bezpečnost, výkonnost a smysluplnost aplikací.[14]

Obrázek 4: Windows 10 logo [15]

3 Technologie pro multiplatformní vývoj

Mimo možnost nativního vývoje pro každou platformu, existuje i multiplatformního vývoj. Tento způsob vývoje je do značné míry, na rozdíl od nativního vývoje výrazně jednodušší, rychlejší a díky jednomu kódu se také dramaticky snižují náklady na vývoj aplikace a její údržbu. Odpadá totiž potřeba existence více vývojářských týmů s výbornými znalostmi každé z platforem. V následující podkapitole budou zmíněny různé možnosti multiplatformního vývoje.

3.1 Xamarin

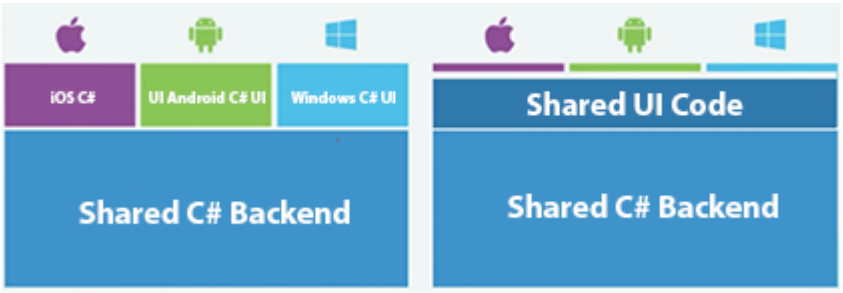
Xamarin je vývojová platforma, která přináší možnost psát nativní multiplatformní aplikace pro tři nejrozšířenější platformy Android, iOS a UWP pro Windows 10. Platformu Xamarin vytvořila stejnojmenná společnost založená v Kalifornii, v květnu 2011. Následně začátkem roku 2016 Microsoft oznámil, že se stal novým a jediným vlastníkem této společnosti.

Vývoj multiplatformních aplikací probíhá pomocí jednoho z nejrozšířenějších programovacího jazyku C# a ve vývojovém prostředí Visual Studio nebo Xamarin Studio.

Samotná platforma Xamarin je cílena pro business nebo náročnější aplikace, kde je kladen důraz na velké množství kódu na pozadí, než v samotné prezenční vrstvě. Jestliže je potřeba vytvořit aplikaci, kde je kladen důraz prakticky pouze na prezenční vrstvu, minimum aplikační logiky nebo požadavek, aby aplikace vypadala na všech platformách stejně, je potřeba zvážit jestli nebude lepší využít jiný nástroj než Xamarin. V opačném případě při kladení důrazu na aplikační logiku, je možné pomocí této technologie dosáhnout velmi dobrých výsledků.[16]

Framework Xamarin se skládá ze třech částí Xamarin.Android (dříve známý jako MonoDroid), Xamarin.iOS (MonoTouch) a Xamarin.Forms. První zmíněnou částí Xamarin.Android, jak už z názvu lze předpovídat, jedná se o framework pro vytváření aplikací pro platformu Android. Stejně tak Xamarin.iOS náleží pro Apple s mobilním operačním systémem iOS. Poslední třetí částí Xamarinu je framework Xamarin.Forms. Toto prostředí dále integruje Xamarin.Android a Xamarin.iOS a prostředí pro Windows 10 (UWP). Kromě business vrstvy a datového modelu umožňuje Xamarin.Forms sdílet také GUI vrstvu, která se stará o práci s obrazovkami a předává data mezi uživatelským rozhraním a business vrstvou.[17]

Detailnějšímu popisu platformy Xamarin bude ještě níže věnována samostatná kapitola.



Obrázek 5: Přístup sdíleného kódu pro framework Xamarin. [18]

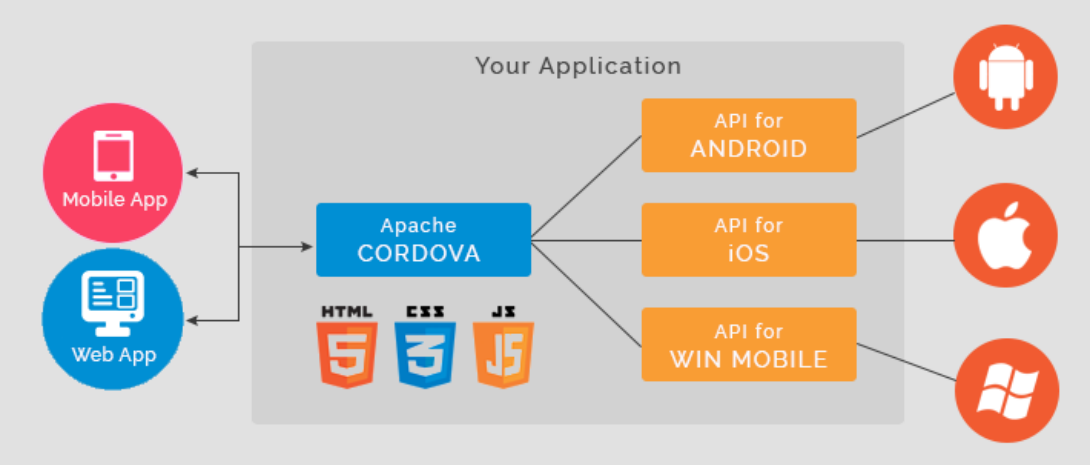
3.2 Apache Cordova

Apache Cordova (dříve nazývaná jako PhoneGap) je open source hybridní framework, určený pro vývoj multiplatformních mobilních aplikací. Umožňuje vytvořit aplikaci za pomoci webových technologií HTML5, CSS3 a Javascriptu. Ve výsledku Apache Cordova obalí vytvořenou aplikaci právě nativní aplikací s webovým prohlížečem, ve kterém se aplikace spustí. Je však důležité, aby nativní aplikace byla již předem zacílená na konkrétní mobilní platformu, aby bylo možné zobrazit obsah, který je vytvořený pomocí webových technologií.

Pro jednodušší vysvětlení si můžeme představit aplikaci, která v podstatě představuje pouze mobilní webové stránky. Následně je pak tento obsah zabalen, zvlášť pro každou platformu do spustitelného balíku. Při spuštění aplikace v telefonu se nejprve spustí komponenta webového prohlížeče, která se v podstatě chová úplně stejně jako mobilní webový prohlížeč. Důležité je však potřeba zmínit, že tato komponenta postrádá hlavní navigační panel s funkcemi pro přechod na novou stránku nebo její obnovení. Následně se pak v této komponentě spustí implementovaná webová aplikace.

Díky připojenému frameworku Apache Cordova má aplikace přístup k nativním službám a funkcím daných platforem. Vývojáři je ulehčena samotná práce a využití nativních API, jako jsou např. přístup k akcelerometru, lokaci, internímu a externímu uložišti a mnoha dalším funkcím.[19]

Velkou výhodou a ulehčením pro vývojáře, je možnost vyvíjet mobilní aplikaci v prostředí Visual Studio a následně spouštět a testovat v běžném webovém prohlížeči. Odpadá tak potřeba jednotlivých emulátorů pro různé platformy, tedy pouze pokud aplikace nepotřebuje přístup k nativním API. Jednoduchý vývoj s sebou přináší i různé nevýhody. Mezi ty největší patří zejména nízký výkon a proto je tento framework pro aplikace s vysokými nároky na CPU z uživatelského hlediska téměř nepoužitelný.[20]

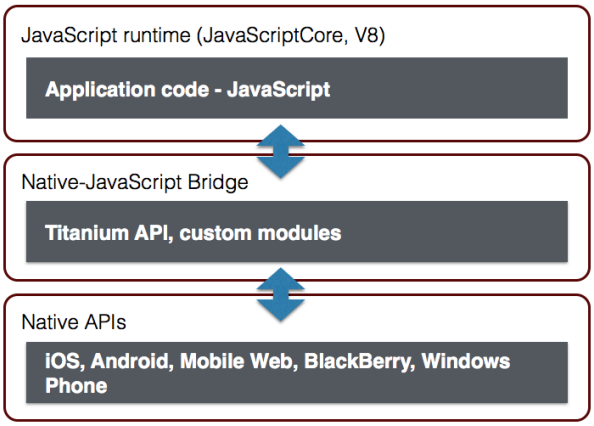


Obrázek 6: Přístupy sdíleného kódy pro framework Apache Cordova. [21]

3.3 Titanium

Produkt Titanium od společnosti Appcelerator je licencován jako open source framework, který umožňuje vytváření nativních, hybridních nebo mobilních webových aplikací, které jsou implementované v objektově orientovaném skriptovacím jazyce Javascript. Pomocí Titania lze vyvíjet na všechny tři největší platformy – Android, iOS a Windows 10 Mobile.[22]

Proces spuštění aplikace probíhá na třech vrstvách, jedná se o vrchní vrstvu pro Javascript, prostřední vrstvu pro framework Titanium a spodní vrstvu pro operační systém dané platformy. Proces aplikace funguje tak, že na Javascriptové vrstvě běží vývojářem vytvořená multiplatformní aplikace, která se následně renderuje do nativní aplikace. Jinými slovy, jestliže vytvoříme např. tlačítko, vytvoří se ve skutečnosti nativní tlačítko v dané platformě. Prostřední vrstva Titanium následně funguje jako most, který propojuje Javasriptovou aplikaci s vytvořeným nativním uživatelským rozhraním. Součástí tohoto mostu je také přístup k funkcím a služeb nativních API daných platforem.[23]



Obrázek 7: Vrstvy přístupu pro framework Titanium. [22]

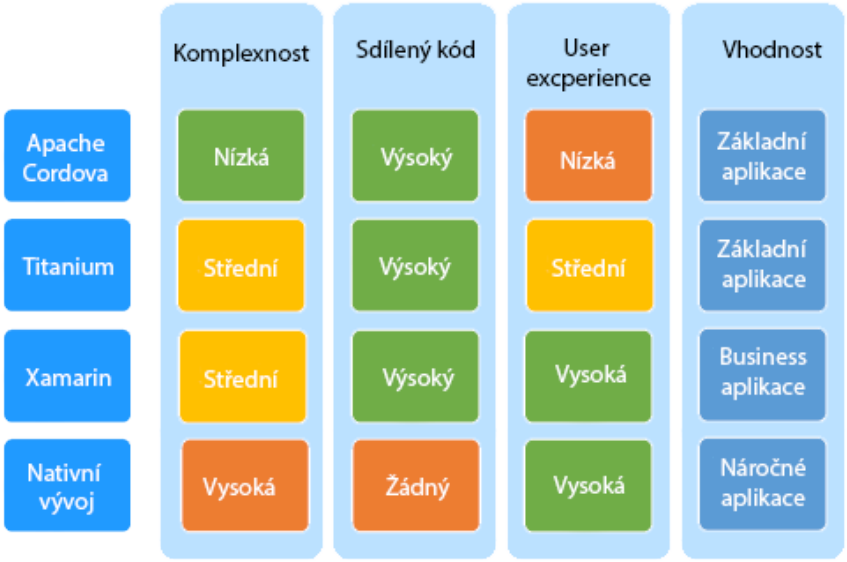
Oficiálním vývojovým prostředím pro tento framework je IDE nazvané Appcelerator Studio. Toto prostřední, stejně jako všechny ostatní nástroje od společnosti, jsou dostupné zdarma. Ovšem pouze do doby, dokud vývojář nepublikuje nějakou aplikaci. Po prvním zveřejnění aplikace je totiž vývojář nucen platit měsíční poplatky společnosti. Tento poplatek se pohybuje okolo 40 amerických dolarů.[24]

3.4 Shrnutí multiplatformního vývoje

Na základě zjištění z různých článků a také vlastních zkušeností jsem vytvořil schéma, které stručně popisuje a porovnává výhody a nevýhody výše zmíněných nástrojů pro multiplatformní vývoj. Ve vytvořeném schématu jsou zachyceny především tyto faktory:

- Komplexnost – popisuje jak těžké a náročné je naučit se vytvářet aplikace pro dané platformy.

- Sdílení kódu – jak velké množství kódu lze sdílet mezi jednotlivými platformami.
- User experience – důležitá vlastnost, která označuje kvalitu výsledné aplikace z pohledu uživatelské použitelnosti a přívětivosti.
- Vhodnost – řeší pro jaké typy aplikací je vhodné využít daný framework.



Obrázek 8: Diagram srovnání multiplatformních nástrojů.

Otázka jaký použít nástroj pro vývoj aplikace, je zcela zásadní. Výběr špatné technologie se totiž může projevit až v budoucnu, např. jako nedostatečná podpora nativních funkcí, špatná stabilita aplikace nebo její nízký výkon. Proto není vždy nejlepší využívat multiplatformní nástroje pro vývoj aplikací, ale je potřeba zvážit všechny možnosti, které závisí vždy na konkrétním projektu.

3.4.1 Multiplatformní vývoj

Výhody

- Nižší náklady na vývoj
- Sdílení kódu skrz všechny platformy
- Rychlejší a jednoduší vývoj

Nevýhody

- Aplikace nemusí být stejně rychlá jako nativní

- Potřeba znalost vybraného multiplatformního řešení
- Omezení některých systémových funkcí

Jestliže uvažujeme o implementaci aplikace, která nebude příliš provázána s hardwarovými prostředky a jejich funkcemi, je vhodné zvážit využití multiplatformního vývoje. Zřejmě nejvhodnějším kandidátem pro univerzální aplikace je nástroj Xamarin, který je velmi blízko nativnímu vývoji. Tento nástroj se hodí pro téměř jakoukoliv aplikaci a navíc lze vyvíjet pro přední mobilní platformy výrazně levněji.

3.4.2 Nativní vývoj

Výhody

- Vysoký výkon
- Vynikající uživatelský dojem
- Plná podpora API zařízení

Nevýhody

- Drahý vývoj pro více platforem
- Potřeba odborných znalostí pro každou platformu
- Vysoké náklady na údržbu

Aplikace založené na nativním vývoji působí lepším uživatelským zážitkem, než některé stávající multiplatformní řešení. Navíc jsou implementovány v jazyku, který je určený pro jejich vývoj, což vede obvykle k nejlepšímu výkonu aplikace. Jestliže nebudeme brát zřetel na cenu vývoje, je tento způsob většinou nejvhodnější.

4 Multiplatformní vývoj v Xamarinu

Jednou z hlavních součástí této práce je implementace multiplatformní aplikace v využití nástroje Xamarin a jak již bylo dříve naznačeno, je potřeba tuto vývojovou platformu více prozkoumat a popsat v samostatné kapitole. Nástroj Xamarin je totiž od ostatních nástrojů pro multiplatformní vývoj natolik odlišný a rozsáhlý, že je třeba věnovat mu více času.

Framework Xamarin je asi nejblíže nativnímu vývoji pro nejrozšířenější platformy, kde lze plně oddělit vývoj společné business logiky a datového modelu od vývoje specifického uživatelského rozhraní zvlášť pro každou platformu. Současně umožňuje i implementování jednoho stejného kódu uživatelské rozhraní pro využití na všech platformách.

4.1 Xamarin.Android

Xamarin.Android slouží pro vývoj aplikací platformy Android. Tento framework využívá pro jeho platformu běhové prostředí založené na projektu Mono, které je licencováno jako open source. Mono je prostředí, které vyvinula společnost Xamarin a jeho hlavním cílem je vytvořit sadu nástrojů, kompatibilní s .NET frameworkem. Současně je Mono postavené na programovacím jazyku C a běží nad linuxovým jádrem společně vedle vlastního běhového prostředí. V praxi se jedná o mapování nativních Android knihoven za účelem multiplatformního vývoje. Tento proces se provádí pomocí malých úprav, které jsou charakteristické pro jazyk C#. Následně tak vzniká vývojáři možnost využití nativních funkcí dané platformy, jako jsou např. uživatelské rozhraní, animace, notifikace nebo také využití specifických funkcí pro geolokaci, gyroskop, fotoaparát a podobně. Současně možnost využití těchto nativních funkcí znamená, že při každé nové vydané verzi operačního systému Android je potřeba, aby byla přizpůsobena i nová verze Xamarinu. Jednoduše řečeno je nezbytné, aby se doimplementovala funkcionalita nového API.[25]

V současnosti má vývojář na výběr dvě možnosti, jakým způsobem bude přistupovat k systémovým prostředkům daného operačního systému. Jednou z možností je využití již právě zmiňované funkcionality standardní .NET knihovny nebo využít systémové API, které je přímo poskytované běhovým prostředím Android a je k dispozici v jazyce C# pomocí tzv. API Binding.

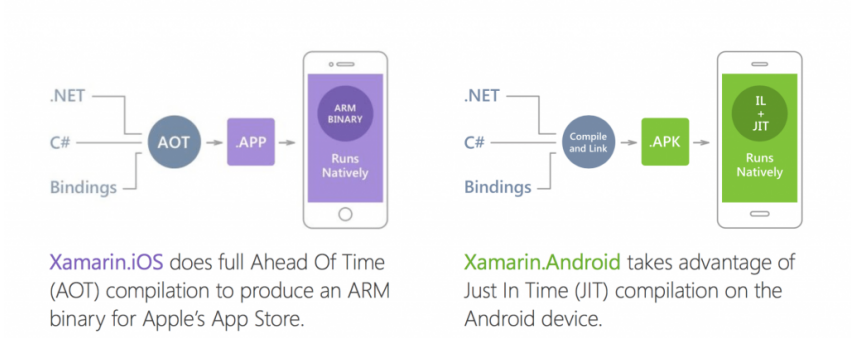
Pro kompilaci Xamarin.Android frameworku je využívána Just-In-Time kompilace. V praxi si můžeme představit C# kód implementované aplikace, který se nejprve zkompiluje do assemblies a ty jsou teprve následně kopírovány a uloženy do výstupního instalačního APK souboru. Při spuštění aplikace dochází k okamžité inicializaci Mono virtuálního stroje, starající se o vykonávání instrukcí, které jsou obsaženy v assemblies.[25]

4.2 Xamarin.iOS

Xamarin.iOS umožňuje vytvářet mobilní aplikace pro operační systém iOS. Stejně jako u Xamarin.Android, tento framework běží na projektu Mono, ale rozdíl nastává právě v kompilátoru kódu. Xamarin.iOS totiž využívá kompilátor zvaný Ahead-of-Time, který přímo překládá apli-

kaci do Assembler kódu. Tento kód je následně využitelný pro procesory, které jsou postavené na ARM architektuře. Součástí kompilace je také překlad všech nativních systémových API funkcí do jazyku C#, které lze dále využívat.[26]

Zároveň možnost kompilace aplikace není každému vývojáři umožněna, je totiž potřeba vyvíjet na zařízení s operačním systémem Mac OS, kde vzniká podmínka vyvíjet v prostředí Xamarin Studio. Druhou možností je vyvíjet v prostředí Visual Studia a mít k dispozici alespoň zařízení od Applu se systémem Mac Os. Pro samotnou kompilaci aplikace vzniká potřeba připojení k systému Mac OS, který umožní vytvoření instalačního balíčku pro aplikaci. V současnosti je také možné ještě provést kompilaci vzdáleně k Mac OS systému a nástroje Build Host Agent. Při samotné kompilaci kódu, jak bylo zmíněno výše, je využívána funkce Ahead-of-Time, která je vzhledem k Xamarin.Androidu spuštěna přímo během kompilace daného kódu.[27]



Obrázek 9: Proces sestavení aplikace pro platformy Android a iOS. [28]

4.3 Xamarin.Forms

Hlavní vlastností tohoto frameworku, je možnost, psát sdílený kód pro různé platformy a sjednotit tak implementaci speciálních funkcionalit, která by se mohla mírně lišit na frameworkem podporovaných platformách. Hlavní myšlenkou je možnost sdílet jeden kód v jednom projektu, pro grafické ovládací prvky, logiku nebo datový model aplikace. Tato velká výhoda přináší následně výrazné urychlení multiplatformního vývoje. V praxi se jedná o soubor tříd a datových struktur, které implementují obecné funkcionality jednotlivých platforem a vytváří nad nimi rozhraní, které následně vývojář využívá v projektu.

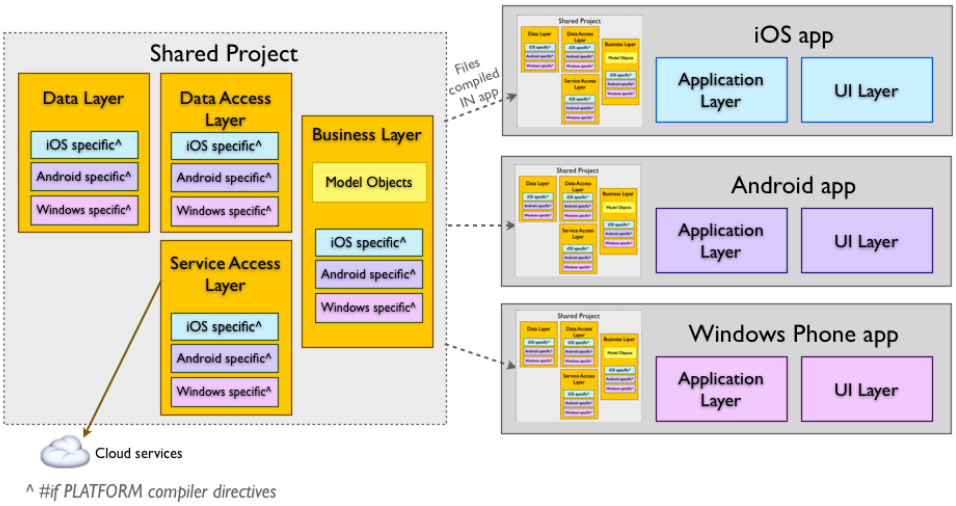
Xamarin.Forms slouží zejména pro vytváření aplikací, které nepotřebují mnoho specifických nativních funkcionalit daných platforem. Následné sdílení kódu umožňuje vývojáři vytvořit jednu funkcionalitu, která je využívána na všech podporovaných platformách. Z pohledu tvorby uživatelského rozhraní, je tento framework skvěle přizpůsoben a jednotlivé grafické komponenty jsou na každé platformě zobrazeny a využívány, přesně podle pravidel platforem pro návrh uživatelského rozhraní.

Současně framework nabízí spousty již vytvořených komponent, co se týče uživatelského rozhraní tak i aplikační logiky. Díky tomuto lze vytvořit aplikaci ve velmi krátkém čase. Nicméně, komponenty můžou mít pro vývojáře často nedostatečné a omezené funkcionality. Tento problém se řeší, vytvořením vlastních funkcí nebo celých komponent, které ovšem můžou přinést zdržení při vývoji.

Struktura tohoto frameworku integruje již zmíněné projekty Xamarin.Android, Xamarin.iOS a také projekt pro univerzální aplikace Windows 10. Hlavním principem frameworku je sdílení kódu pro jednotlivé projekty. Tímto se rozumí sdílení datového modelu, business vrstvy a také aplikační vrstvy, která má na starosti především práci s obrazovkami, jejich navigaci a předávání dat mezi jednotlivými vrstvami. Tento kód pro sdílení lze implementovat ve třech variantách a to v knihovně Shared Code, Portable Class Library (PCL) a nebo v knihovně .NET Standard.

4.3.1 Shared Project

První varianta knihovny pro sdílení kódu, kterou využívá framework Xamarin.Forms je Shared Code projekt. Nejedná se sice o plnohodnotný projekt, ale pouze o "obálku", která má za úkol obalit různé soubory z .NET technologie, jako jsou soubory s příponou .cs, .xaml, .resx a podobně. Při kompilaci následně dochází k tomu, že se soubory tváří jakoby byly přímo součástí projektů, které referencují Shared Code knihovnu. Z tohoto vyplývá nutnost referencovat v každém kompilovaném projektu všechny závislosti, které jsou použity v souborech ve sdílené knihovně. Díky těmto povinnostem projekt neobsahuje žádné další speciální soubory, jako jsou v jiných projektech, ale pouze soubory, které se vytvoří a přidají při implementaci aplikace.[29]



Obrázek 10: Náhled struktury projektu s Shared knihovnou. [29]

Další vlastností Shared Code projektu je možnost oddělení implementace pro různé platformy. Jestliže je např. potřeba v některých třídách oddělit kód mezi Androidem a iOS, lze využit tzv. direktiva kompilátoru. Tato direktiva zajistí, že v době kompilace kódu se využije

pouze část kódu, která je určena pouze pro danou platformu. Tuto možnost rozlišování kódu lze používat i v dalších sdílených knihovnách, ale nejčastější využití je právě v projektu Shared Code.[30]

```
public static string FilePath
{
   get
   {
      var path = string.Empty;
      #if __ANDROID__
         path = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
      #else if __IOS__
         path = Environment.GetFolderPath (Environment.SpecialFolder.Personal);
      #else // UWP
         path = Windows.Storage.ApplicationData.Current.LocalFolder.Path;
      #endif
   }
}
```

Výpis 1: Použití direktiv pro získání cesty ke sdílené složce každé platformy.

4.3.2 Portable Class Library

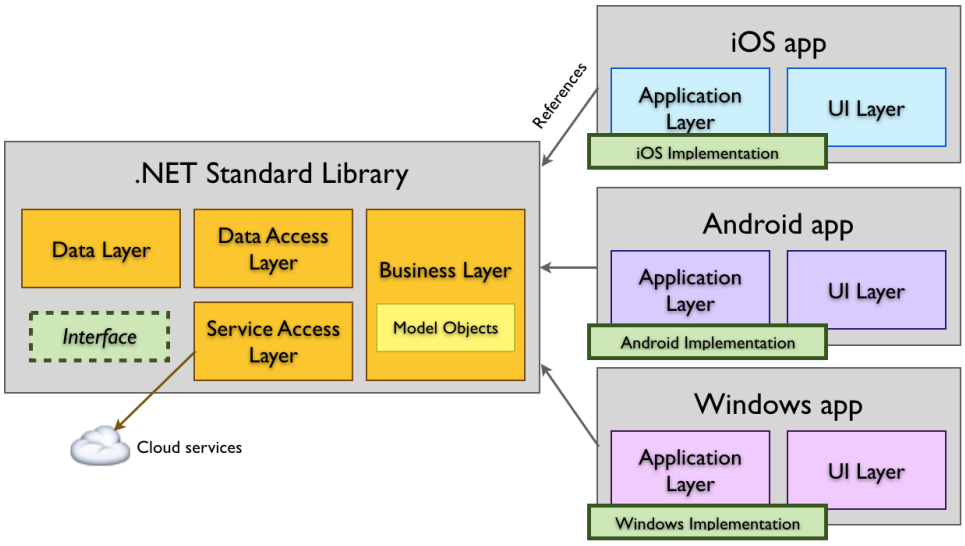
V současné době zatím ještě nejpoužívanější knihovna pro sdílení kódu mezi platformami. Portable Class Library je typ knihovny, který umožňuje vývojáři psát sdílený kód za pomocí omezeného .NET frameworku (Base Class Libraries) pro vybrané platformy. Mezi standardní platformy, které jsou v Portable Class Library k dispozici patří .NET, UWP Windows 10 a Silverlight. Ve skutečnosti při vytváření PCL knihovny si sami zvolíme, které platformy je třeba podporovat a podle toho se automaticky vybere nejvýhodnější podmnožina .NET frameworku. V praxi to znamená, že čím větší počet platforem potřebujeme podporovat, tím menší podmnožinu .NET frameworku máme k dispozici. Automaticky s vytvořením PCL projektu se vytvoří i projekty pro Xamarin.Android, Xamarin.iOS a UWP pro Windows 10.[31]

Při využívání knihovny Portable Class Library vývojáři odpadají problémy vznikající s podmíněným překladem. Závislá funkcionalita mezi PCL knihovnou a projekty je zde dodána pomocí Dependency Injection. Jedná se o vkládání závislostí mezi jednotlivé projekty tak, aby jeden projekt mohl používat jiný, aniž by měli při kompilaci referenci na sebe. Mezi hlavními nevýhody PCL knihovny je chybějící plnohodnotný .NET framework. Naopak tato nevýhoda přináší i velkou výhodu a tou je možnost využití systému Nuget pro stahování a instalování knihoven třetích stran a doplnění tak chybějící potřebné funkcionality z .NET frameworku. V současné době obsahuje Nuget manažer stovky knihoven, které lze využít v projektu PCL.[31]

4.3.3 .NET Standard

Tento projekt je nejmladší knihovnou pro sdílení kódu napříč jednotlivými platformami. Jedná se o knihovnu vytvořenou Microsoftem, která pokrývá větší funkcionality z .NET frameworku, než knihovna Portable Class Library, a proto se také předpokládá, že by měla zřejmě zcela nahradit tuto knihovnu.

Knihovna .NET Standard je především formální specifikace .NET API a je dostupná ve všech .NET běhových prostředí. V praxi to znamená, že knihovna napsaná pomocí této technologie může být použita v libovolném projektu v .NET frameworku, jako je např. Xamarin, .NET Core nebo WPF. Tímto může vývojář ve svých aplikacích dramaticky zvýšit použitelnost již vytvořených knihoven v .NET frameworku.[32]



Obrázek 11: Náhled struktury projektu s knihovnou .NET Standard. [33]

4.4 Možnosti vývoje

Pro platformu Xamarin lze momentálně vyvíjet ve dvou různých vývojových prostředích a to Xamarin Studio a Microsoft Visual Studio. Nástroj Xamarin Studio je především přizpůsoben k vývoji aplikací pro systémy Android a iOS, při využití nástroje Visual Studia lze tyto možnosti ještě rozšířit i pro vývoj univerzálních aplikací pro Windows 10. V následující tabulce jsou zaznačeny různé možnosti vývoje na základě různých kombinací vývojového prostředí a operačního systému.

	Mac OS X	Windows	
Vývojové prostředí	**Xamarin Studio**	**Visual Studio**	**Xamarin Studio**
Xamarin.iOS	Ano	Ano (s připojením k Mac)	Ne
Xamarin.Android	Ano	Ano	Ano
Xamaarin.Forms	iOS A Android	Android, iOS, UWP Windows	pouze Android

Obrázek 12: Tabulka multiplatformních přístupů v Xamarinu pro operační systémy.

Na základě této tabulky lze prohlásit, že možnost vývoje pro všechny podporované platformy je možný pouze při kombinaci prostředí Visual Studio a operačního systému Windows s přístupem síťového připojení k zařízení s Mac OS.

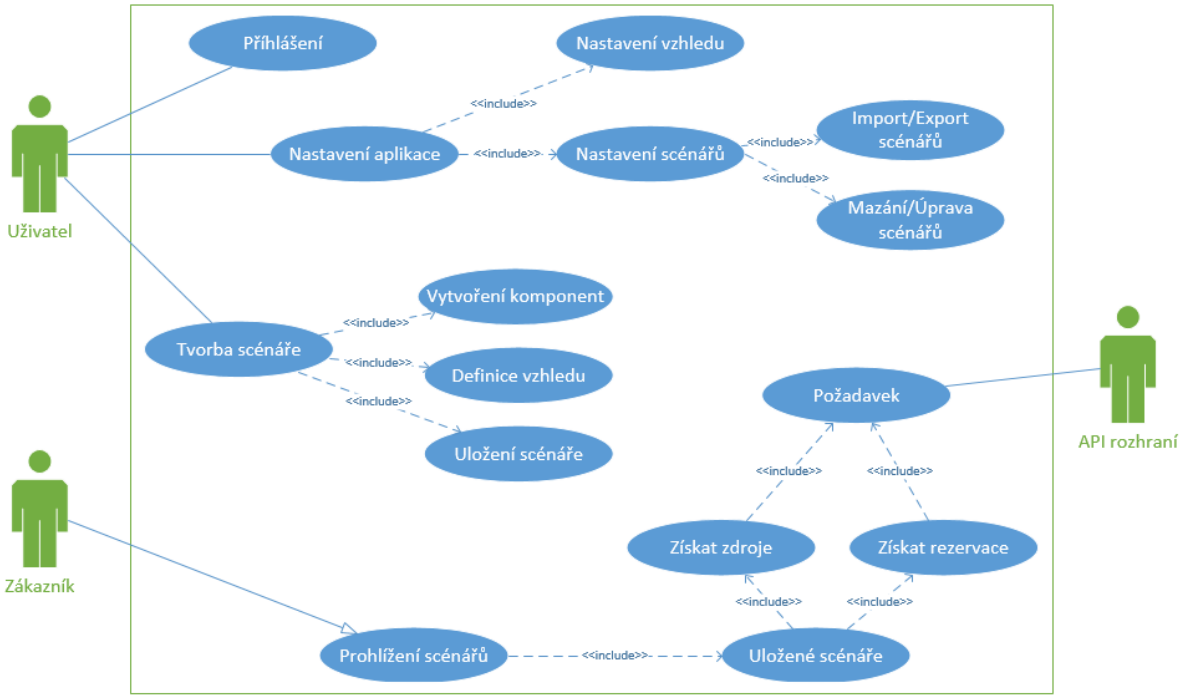
5 Případová studie vývoje multiplatformní aplikace

Součástí této práce je vývoj aplikace pomocí zvoleného nástroje pro multiplatformní vývoj. V této kapitole se budu dále zabývat prvotní analýzou aplikace a jejími specifikacemi, které je důležité nastavit před každým začátkem vyvíjeného systému nebo aplikace. Jedná se především o stanovení cílů, popis aplikace, funkční a nefunkční požadavky a neposlední řadě i výběr technologie pro implementaci.

5.1 Specifikace cíle

Jedním ze základních cílů této práce bylo prostudování jednotlivých možností multiplatformního vývoje a výběr vhodného frameworku pro vytvoření aplikace, která bude spustitelná na více platformách. Po prostudování jednotlivých možností, jsem vyhodnotil jako nejlepší pro mou implementaci framework Xamarin.Forms. Hlavním aspektem pro výběr tohoto nástroje byla možnost využití značného množství sdíleného kódu a současně využití nativního vzhledu jednotlivých grafických prvků pro dané platformy.

Pro zachycení specifikace cíle byl vytvořen UML diagram případu užití. Jedná se o zachycení základních funkcionalit aplikace pro internetový rezervační systém. Je zde zachycen uživatel a zákazníci jako aktéři systému. Uprostřed jsou znázorněny případy užití, které aplikace obsahuje. A na pravé straně je zaznačen externí systém představující API rozhraní pro výměnu dat.



Obrázek 13: UML Use case diagram pro mobilní aplikaci.

5.2 Popis aplikace

Aplikace bude sloužit jako prostředník mezi zákazníkem a uživatelem rezervačního systému. Základní funkcí aplikace bude práce se zarezervovanými zákaznickými úkoly a jejich následné zobrazovaní v různých podobách. Zobrazované rezervační úkoly aplikace budou k dispozici přes požadavek na API rozhraní, ze kterého jsou odesílány data na základě příchozích parametrů požadavku. Aplikaci je potřeba optimalizovat pro LCD panely s velkou úhlopříčkou a podporou operačního systému.

Součástí aplikace bude seznam komponent pro jejich výběr a rozložení do tzv. scénářů, které se budou zobrazovat zákazníkům na hlavní obrazovce. Komponenty budou představovat jedno zobrazení daných rezervací v určitém uskupení, např. zobrazení právě probíhajících nebo následujících rezervací, zobrazení počtu zpracovaných rezervací, zobrazení jednoduchého textu nebo multimediálního obsahu atd.. Tyto komponenty bude možné následně uspořádat do scénářů, které představují rozvržení stránky, např. bude možné přidat jednu komponentu do levé části stránky a dvě komponenty pod sebe do pravé části stránky nebo např. přidat pouze jednu komponentu na celou stránku atd.. Následně se takovému scénáři přiřadí čas, po který se má daný scénář na hlavní obrazovce zobrazovat, po vypršení této doby přichází na řadu další scénář a tento proces se neustále opakuje.

5.3 Specifikace aplikace a její požadavky

Požadavky na aplikaci dělíme na dva základní typy, jedná se o funkční a nefunkční požadavky. Funkční požadavky si můžeme představit jako požadavky, kde je kladen důraz především na věcný a problémový obsah daného systému. Jednoduše řečeno je potřeba definovat co má aplikace obsahovat. Naopak nefunkční požadavky jsou požadavky softwarové architektury, které kladou důraz především na proces vývoje, jeho provedení a na celkový systém. Tím rozumíme především požadavky na výkonnost a vzhled aplikace neboli jakým stylem a způsobem by měla být aplikace implementována. Obecně platí, že při vývoji mobilních aplikací jsou funkční a nefunkční požadavky mírně odlišné, než při vývoji klasických desktopových nebo webových systémů.[34]

5.3.1 Nefunkční požadavky

- **Operační systém**
  Primárně bude aplikace využívána na platformě Android (verze 4.4 a výše), ale do budoucna se předpokládá, že bude zapotřebí, aby byla možnost aplikaci spustit také na platformě iOS a Windows 10 Mobile.

- **Zdroj dat a způsob komunikace**
  Aplikace bude komunikovat s webovým API rozhraním rezervačního systému. Komunikace bude probíhat za pomocí HTTP požadavku GET a POST, který může obsahovat v hlavičce specifické data, důležité pro upřesnění daného požadavku.

- **Zpracovávání požadavků**
  Je nezbytně nutné, aby dlouhotrvající požadavky a výpočetní operace aplikace neblokovaly uživatelské rozhraní. Z tohoto důvodu je potřeba, aby aplikační logika byla implementována pomocí asynchronních metod. V desktopových a webových systémech je toto řešení pouze doporučené, naopak v mobilních aplikacích je považováno za nezbytné.

- **Typ přenášených dat**
  API rozhraní bude zpracovávat jednotlivé požadavky aplikace a následně vyhodnocenou odpověď bude odesílat zpět pomocí datového řetězce pro výměnu dat ve formátu JSON.

- **Architektura a vzhled aplikace**
  Tvorba aplikací pro jednotlivé mobilní platformy má mírně odlišené zásady. Nejvíce se tyto zásady projevují na uživatelském rozhraní, protože každá platforma pracuje různým způsobem s dostupnými prostředky uživatelského rozhraní. Pro aplikaci je tak důležité, aby se uživatelské rozhraní co nejvíce podobalo zvyklostem jednotlivých platforem. Z tohoto důvodu je potřeba využít framework Xamarin.Forms, aby uživatelé byli schopni jednoduše a intuitivně používat aplikaci.

  Aplikace bude zároveň využívána pouze na velkých obrazovkách nebo monitorech, tudíž není potřeba optimalizovat zobrazení pro menší obrazovky než 10 palců.

- **Případná nedostupnost k internetu**
  Obecně u mobilních aplikací platí, že problém s nedostupností k internetu nastává velmi často a je potřeba, aby aplikace si s tímto problémem poradila. Z tohoto důvodu je nutné aplikaci navrhnout a implementovat tak, aby si dokázala poradit s možnými výpadky v průběhu používání aplikace. V rámci této problematiky, bude aplikace zobrazovat poslední aktualizované data s datum poslední aktualizace se serverem.

- **Zpracování výjimek**
  Aplikace by neměla v případě chyby zobrazovat interní hlášení o dané chybě, které je pro obyčejné uživatele velmi matoucí. Tento druh hlášení bude zaznamenán na interním úložišti u instalace aplikace v log souboru. V případě výskytu takového problému bude aplikace nadále standardně použitelná.

- **Podpora více jazyků**
  Vzhledem k různo-jazyčnosti uživatelů rezervačního systému, vzniká potřeba podpory více jazyků, které by měly být v aplikaci podporovány. V současnosti jsou zapotřebí pouze dva jazyky, výchozí jazyk anglický a český jazyk.

5.3.2 Funkční požadavky

- **Aktuální a následující rezervace**
  Hlavní myšlenkou aplikace je zobrazování rezervací vytvoření v rezervačním systému. Tyto

rezervace je potřeba zobrazovat v jednoduchých a přehledných uskupení, které usnadní zákazníkům orientaci u uživatele. Uživatel představuje obchodníka nebo společnost, která využívá rezervační systém pro své podnikání a zákazník označuje osobu objednávající si službu u daného obchodníka nebo společnosti.

- **Přehled rezervací**
  Mimo zobrazování jednotlivých rezervací by měla aplikace také umět zobrazit rezervace na celý den. Toto zobrazení by mělo klást důraz především na jednoduchost a přehlednost rezervací.

- **Sumarizace rezervací**
  Pro rychlý přehled je potřeba zobrazovat aktuální stav rezervací. Jde především o přesný počet již proběhlých rezervací a také počet rezervací, které jsou ve stavu, kdy čekají teprve na vyřízení. Zobrazením těchto rezervací je potřeba ještě rozlišovat mezi součtem pouze pro určitý zdroj a součtem pro všechny zdroje. Tyto zdroje se nacházejí v samotných rezervačních systémech a rozdělují se většinou podle typů jako je místo, služba a zaměstnanec. Následně pak uživatel může vytvářet úkoly s určitou kapacitou k těmto zdrojům a jejich zákazníci si tyto místa můžou rezervovat.

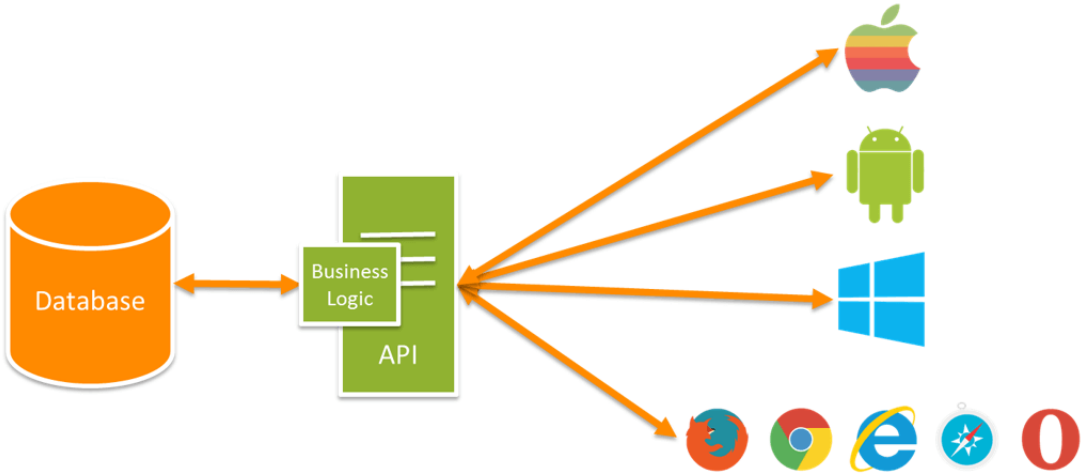
- **Text, datum a čas**
  Součástí aplikace je potřeba zobrazovat v některých případech datum a čas, který je potřeba, aby byl volitelný v několika různých formátech. Současně je také potřeba zobrazovat jednoduchý text pro libovolné uživatelské informace.

- **Multimediální obsah**
  Možnost zobrazovaní obrázků a videa pomocí hypertextového odkazu bude sloužit jako mírné oživení aplikace a současně také např. jako uživatelská reklama pro jejich zákazníky.

- **Autentizace uživatele**
  Z důvodů snadnějšího přístupu a spouštění aplikace je potřeba, aby si aplikace dokázala zapamatovat uživatelské přihlašovací údaje a přístup k API rozhraní.

5.4 API – napojení

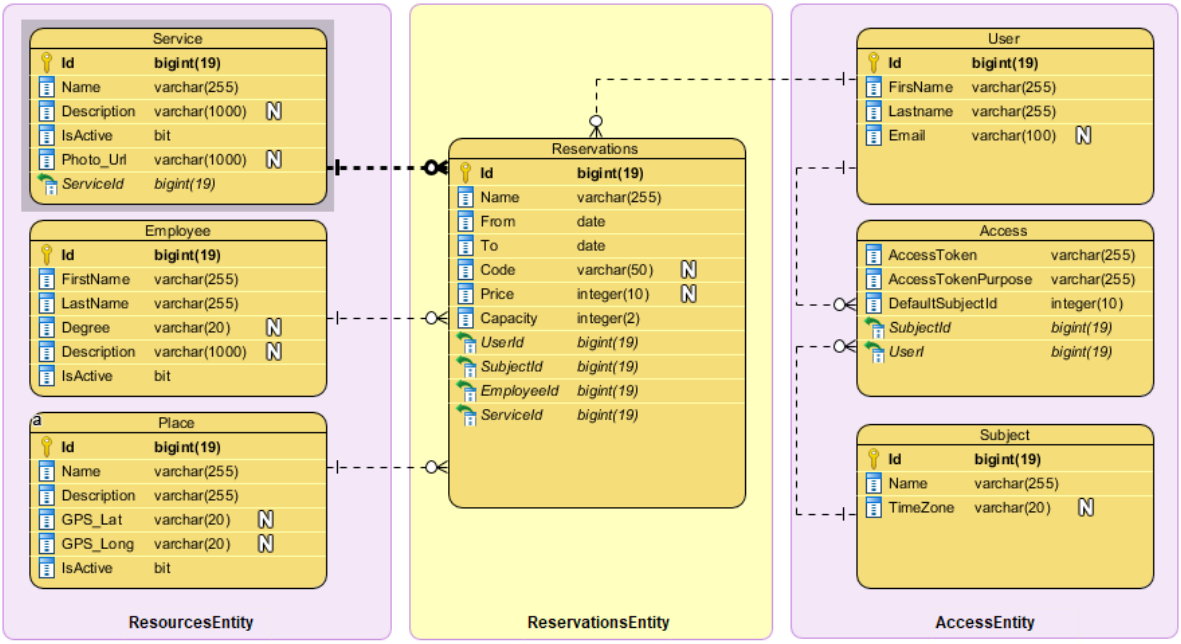
Pro komunikaci mezi implementovanou aplikací a rezervačním systémem, bude využito rozhraní API. Pro zabezpečení odesílání dat z API rozhraní slouží tzv. přístupový token. Veškerá komunikace aplikace s API rozhraním bude probíhat právě pomocí tohoto tokenu. Jestliže API přijme nevalidní token, data nebudou odeslána zpět na zařízení odkud přišel požadavek, ale bude odeslán pouze chybový kód. Validní token lze získat pomocí správných přihlašovacích údajů (e-mail a heslo) do rezervačního systému, které jsou odeslány požadavkem z aplikace na API rozhraní. Na základě tohoto přístupového tokenu bude aplikace muset zpracovat a zobrazit rezervační úlohy, které budou odesílány pomocí řetězce JSON.



Obrázek 14: Výměna dat skrz API rozhraní. [35]

5.5 Datový model

Datový model zobrazuje datovou strukturu, která bude využívána pro práci s konkrétními daty s nimiž pracuje informační systém. Datový model popisuje formát a strukturu dat v systémech a současně určuje vzájemné vztahy mezi jednotlivými třídami neboli entitami. Cílem datového modelu je právě zachytit a popsat tu část reality, o které chceme ukládat informace. Tuto realitu pak následně transformuje do již zmíněných entit s jednotlivými vazbami mezi sebou.[36]



Obrázek 15: Datový model přenášejících dat. [35]

Na obrázku lze vidět datový model, který bude využíván v implementované aplikaci pro mapování dat z rezervačního systému na základě příchozího JSONu ze zpracovaného požadavku. Jedná se o pět entit, které reprezentují tyto databázové tabulky:

- **ReservationsEntity** – zde jsou zaznamenány veškeré rezervace, které jsou vytvořeny zákazníkem a jsou uloženy v databázi.
- **ResourceServicesEntity** – obsahuje uživatelovy zdroje, které jsou označeny jako služby. Pro příklad si můžeme představit uživatele, který provozuje fitness centrum. Služby zde mohou představovat jednotlivé kurzy fitness trenérů.
- **ResourceEmployeesEntity** – v tomto typu zdrojů se budou zobrazovat zaměstnanci. V příkladu u fitness centra by zde byli zapsáni všichni fitness trenéři.
- **ResourcePlacesEntity** – poslední zdroj představuje místa. Zde si můžeme představit jednotlivé místnosti ve kterých se budou odehrávat služby s daným zaměstnancem.
- **AccessEntity** – tato entita představuje obraz uživatele rezervačního systému. Je zde zaznamenán název subjektu, popřípadě jméno a příjmení a podobně.

5.6 Logická vrstva aplikace

Důležitou součástí aplikace je oddělení logické vrstvy aplikace od datového modelu a uživatelského rozhraní. Jedná se především o logickou část aplikace, která se soustředí na logické operace a výpočty neboli jednoduše řečeno představuje "mozek"celé aplikace.

V implementované aplikaci bude tento princip řešen pomocí návrhového vzoru MVVM, který odděluje logiku aplikace od uživatelského rozhraní. Tento vzor bude detailněji popsán v pozdější kapitole, která se přímo věnuje použitým návrhovým vzorům v aplikaci.

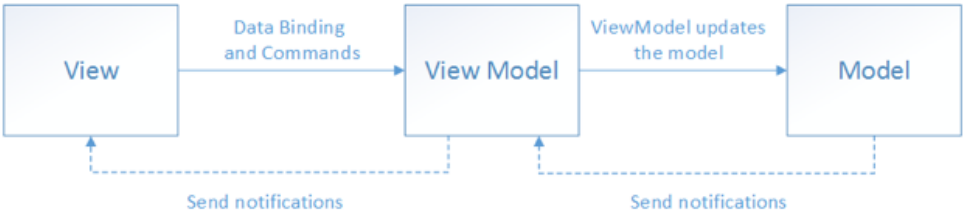
6 Návrh aplikace

V této kapitole se budu zabývat návrhem aplikace především z implementačního pohledu. V první řadě popíšu několik návrhových a architektonických vzorů, které jsem využil při návrhu aplikace. Návrhovým a architektonickým vzorem se řešení daná problematika v různých situacích při vývoji. Typicky se jedná o uskupení tříd, objektů a vztahů mezi nimi, podle určených pravidel. Dále jsou v kapitole popsána témata zabývající se použitou technologií při vývoji aplikace a v neposlední řadě také návrh uživatelského rozhraní a specifické funkce aplikace.

6.1 Použité návrhové a architektonické vzory

6.1.1 MVVM model

Model-View-ViewModel, zkráceně MVVM, je označení návrhového vzoru pro grafické aplikace. Tento model nabízí řešení, jakým způsobem oddělit logiku aplikace od uživatelského rozhraní. Způsob implementace je vhodný zejména, protože lze vyvíjet uživatelské rozhraní a logiku aplikace odděleně třeba i pomocí více vývojářů. [38]

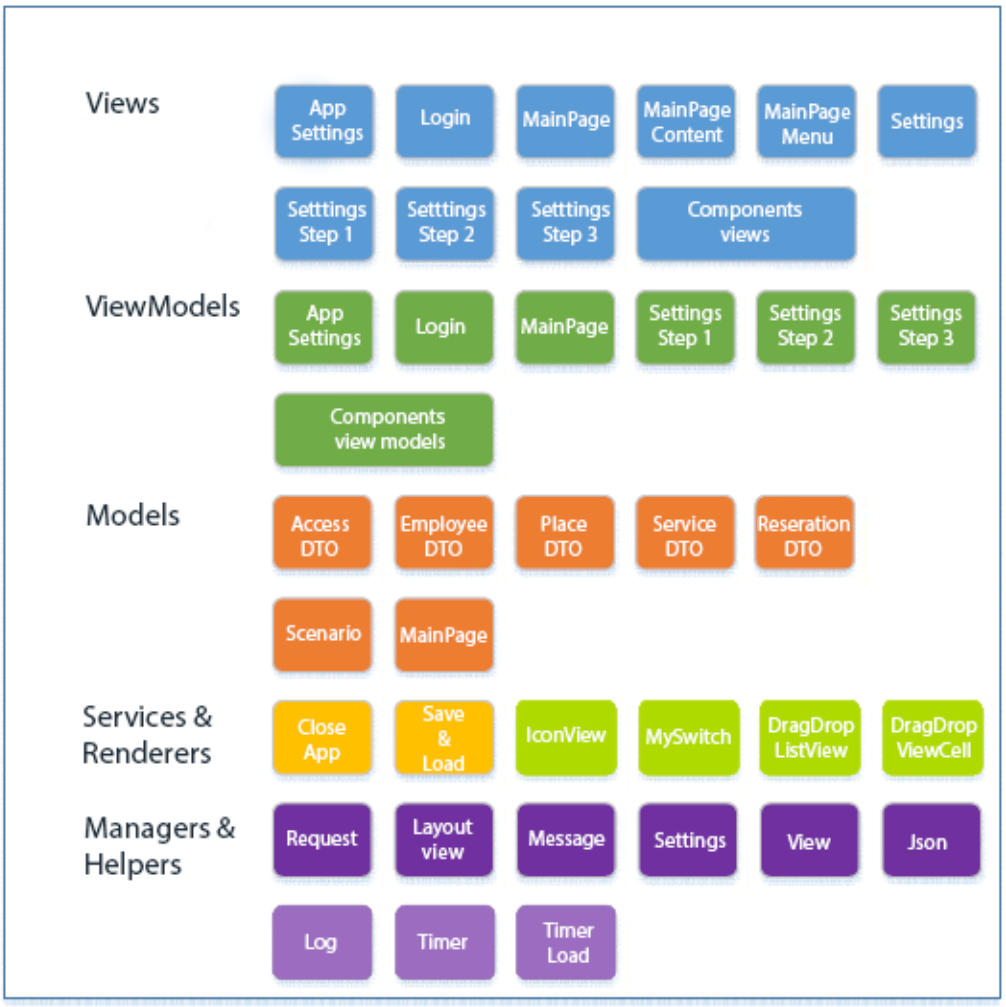


Obrázek 16: Struktura komunikace v MVVM modelu. [37]

Hlavní myšlenkou MVVM je vytvořit třídu, která si udrží aktuální stav aplikace. Toto představuje třída ViewModel. Uživatelské rozhraní následně komunikuje s touto třídou a na základě jejich vlastností vykresluje grafické ovládací prvky. A naopak, jestliže se změní uživatelské rozhraní z pohledu uživatele, přenesou se tyto informace automaticky do ViewModelu. ViewModel je nejdůležitější třídou, která poskytuje veškerá data pro uživatelské rozhraní, které se nazývá View. Veškerá data jsou poskytována za pomocí datových struktur, které dokáží při jejich změně vyvolat událost a tím změnit uživatelské rozhraní neboli View. Jedná se o dva základní prvky. První z nich je kolekce ObservableCollection<T>, která informuje o změně prvku v kolekci. Druhý prvek je rozhraní INotifyPropertyChanged. Při použití tohoto rozhraní je potřeba implementovat i událost, která se vyvolá, jestliže se změní nějaká vlastnost ViewModelu. Tuto událost automaticky odebírá View daného ViewModelu a aktualizuje tak potřebné grafické prvky. Součástí MVVM vzoru je i třída Model, která obsahuje data, se kterými aplikace pracuje.[37][38]

Zobrazení MVVM modelu a složení aplikace lze zachytit pomocí aplikačního diagramu, který je zobrazen na následujícím obrázku. Hlavním principem diagramu je zachycení aplikačních tříd a jejich kategorizace do příslušných vrstev. Diagram zobrazuje pouze třídy pro sdílený projekt a

neobsahuje žádné třídy z použitých platformních projektů. V následujících podkapitolách bude popsán detailní popis jednotlivých vrstev.



Obrázek 17: Rozložení tříd a složek v projektu PCL.

6.1.1.1 Views Třída View reprezentuje uživatelské rozhraní pro danou platformu a je implementována v jazyce XAML s tzv. code-behind (kód na pozadí) v C#. Veškeré uživatelské rozhraní by mělo být implementováno právě pomocí jazyka XAML a na pozadí v C# kódu by měla být pouze v ojedinělých případech jednoduchá implementace. Třídy se v ideálním případě vyskytují ve vztahu 1:1 k ViewModelům. V aplikaci se View může vyskytovat jako ovládací prvek, stránka nebo také celé okno aplikace. Pomocí View můžeme označovat také uživatelské rozhraní, formulář nebo také prezentační vrstvu.[37]

V aplikaci budou implementovány Views ve sdílené PCL knihovně a pro žádnou platformu nebudou implementované nativní uživatelské rozhraní. Implementované Views můžeme vidět na obrázku 17, který zobrazuje aplikační diagram.

6.1.1.2 ViewModels

ViewModel, jak už bylo naznačeno, obsahuje definice, co se má zobrazovat na dané obrazovce. Jedná se především o property, kolekce nebo commandy (akce které je možné vykonávat na obrazovce). ViewModel slouží jako prostředník mezi Modelem a View, který si drží svůj stav. Komunikace mezi jednotlivými vrstvami následně probíhá pomocí bindingu, kdy jsou data přenášeny jak jednosměrně tak i obousměrně.[37]

Podle MVVM vzoru lze vidět na obrázku 17 návrh ViewModelů, které budou implementovány v aplikaci. Podle pravidel se jedná o třídy 1:1 k Views neboli ke každému View bude existovat maximálně jeden samostatný ViewModel.

6.1.1.3 Models

Jak již bylo nastíněno, jedná se o vrstvu s třídami, které popisují datovou strukturu používanou v aplikaci. Každá tato třída nějakým způsobem popisuje data, kdy je většinou její struktura shodná s tabulkami v databázi, z důvodu lepší orientace při vývoji. Základní vlastností třídy je pravidlo, že Model nesmí vědět nic o stavu ovládacích prvků.[37]

V této vrstvě budou implementovány třídy, které slouží jako šablona pro datovou strukturu použitou pro zobrazování dat v aplikaci. Jedná se zejména o třídy ScenarioModel, MainPageModel a další.

6.1.2 Publish - Subscriber

Publish-Subscriber je návrhový vzor určený především pro rozesílání zpráv mezi třídami. Ve frameworku Xamarin.Forms je tento vzor reprezentován pomocí třídy MessagingCenter. Tato třída obsahuje dvě základní části a to:

- Subscribe – metoda, pomocí které se lze přihlásit k odběru rozesílaných zpráv na základě určitého klíče.
- Send – metoda, která publikuje zprávu s konkrétním klíčem, ke kterému se mohou ostatní objekty přihlásit.

Princip tohoto vzoru spočívá v tom, že se nejprve v dané třídě, ve které je potřeba zprávu přijímat, zaregistruje odběr dané zprávy pomocí statické metody Subscribe z třídy MessagingCenter. Následně se zpráva, kterou chceme posílat nadefinuje a pomocí statické metody Send se automaticky odešle všem zprávám, které mají definován odběr zprávy odesílané se stejným klíčem.[39]

Mezi hlavní výhody tohoto vzoru patří zejména odstranění závislostí mezi zdrojem a příjemcem jako je tomu u klasických události. Současně lze tyto události (Event) nahradit pomocí této třídy MessagingCenter a vyhnout se tak problémům (jako jsou tzv. memory leaky), které v klasických událostech mohou vznikat při neopatrné manipulaci s událostmi. Naopak hlavní nevýhodou je zpožděné doručování zpráv, jelikož takto odesílané zprávy nemusí být doručovány okamžitě.

```
// sender
MessagingCenter.Send<string>(AppResources.MSG_SAVE, MsgEnum.
    AllertToAppSettingView.ToString());
// subscriber
MessagingCenter.Subscribe<string>(this, MsgEnum.AllertToAppSettingView.ToString
    (), (arg) => {
DisplayAlert(AppResources.LBL_INFORMATION, arg, AppResources.LBL_OK);
});
```

Výpis 2: Ukázka kódu pro zasílání a přijímání zprávy.

Tento návrhový vzor bude v implementované aplikaci použit jako komunikátor mezi View-Modelem a View především pro zobrazení pop up dialogů, které nelze vyvolat na logické vrstvě aplikace ale pouze na vrstvě uživatelského rozhraní. A právě pro tuto problematiku je tento návrhový vzor ideálním pomocníkem, jak posílat informaci o nějaké změně.

6.1.3 Singleton

V aplikaci je potřeba tento návrhový vzor použít hned v několika třídách. Ukázkovým příkladem může být např. využití v navigační třídě, která se stará o zobrazování a přepínání jednotlivých obrazovek. Jelikož celá aplikace bude vytvořena pomocí několika obrazovek, kde bude potřeba udržovat jejich posloupnost a volat specifickou logiku při jejich zobrazovaní a přepínání, nebylo by možné v každé třídě vytvářet novou instanci navigační třídy a zároveň by nebylo ani praktické si jí stále přeposílat parametrem konstruktoru.

```
public class IdManager
{
   private static IdManager _instance = null;
   private IdManager() { }

   public static IdManager Instance {
      get
      {
         if (_instance == null)
            _instance = new IdManager();
         return _instance;
      }
   }
}
```

Výpis 3: Implementace návrhového vzoru Singleton.

6.1.4 Strategy

Návrhových vzor Strategy je vzor, který umožňuje výměnu různých implementací za běhu aplikace bez nutnosti změny kódu programu. Tento vzor může být tvořen abstraktní třídou nebo rozhraním, které se nazývá Strategy. Tato třída nebo rozhraní definuje předpis, jakým způsobem mají být implementovány jejich potomky, kteří si následně implementují svou vlastní logiku.[40]

Tento návrhový vzor bude v aplikaci použit při vytváření obrazovek pro jednotlivé komponenty. Definováním jednoho stejného rozhraní pro všechny View a také jednoho pro všechny jejich modely, zajistíme, že při vytváření instancí a jejich následné komunikaci, bude zajištěna správná funkcionalita dané komponenty.

6.2 Použité technologie

Součástí návrhu aplikace je potřeba definovat technologie, které budou využívány při její implementaci. Jak již bylo několikrát zmíněno, pro multiplatformní vývoj aplikace jsem vybral framework Xamarin.Forms, pro jeho výhody, které byli také zmíněny v dřívějších kapitolách. V příloze A této práce budou popsány programovací jazyky XAML a C#, které jsou základním kamenem frameworku Xamarin.Forms.

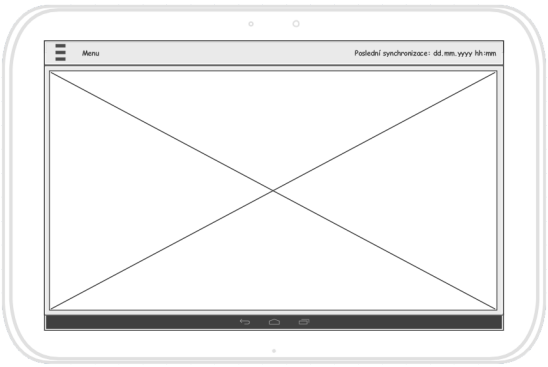
6.3 Návrh uživatelského rozhraní

Jedním z požadavků na aplikaci je její multiplatformnost a proto je potřeba klást důraz na návrh uživatelského rozhraní před samotným vývojem aplikace. Jelikož je potřeba, aby aplikace podporovala tři různé druhy platforem, je nutné brát zřetel na jejich odlišnosti v uživatelském rozhraní. Jde zejména o zamezení využití specifických grafickou prvků, které jsou podporovány pouze na některých platformách. Následně tak nebude potřeba vyvíjet grafické prvky zvlášť pro platformy, které je neobsahují. Tím docílíme minimalizace kódu, který není sdílený pro všechny platformy, ale jen cílený pouze na určitou platformu. Framework Xamarin.Forms obsahuje průnik všech grafických prvků, které podporují všechny jeho platformy. Je-li potřeba prvek, který zde není, musí si jej programátor do implementovat zvlášť pro každou platformu.

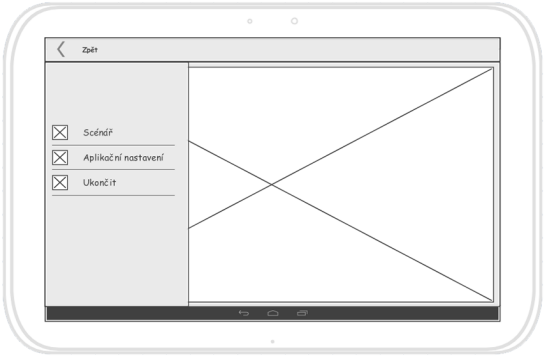
Z důvodů velkého množství obrazovek v aplikaci jsou v následujících podkapitolách popsány pouze vybrané obrazovky uživatelského rozhraní a jejich funkcionalita. Popis obrazovek bude popsán pomocí wireframů, které slouží právě pro jejich náčrt.

6.3.1 Hlavní obrazovka

Hlavní obrazovka bude sloužit pouze jako informační prvek, kde budou zobrazeny scénáře, které si uživatel aplikace může sám vytvořit. Co jsou scénáře a jak fungují, bude popsáno později v samostatné sekci. Tato obrazovka bude implementována jako MasterDetailPage nebo-li bude z ní možné přejít pomocí navigace do menu a dalších obrazovek aplikace. Tlačítko pro zobrazení menu bude jediný uživatelský prvek pro akci a zobrazí se v případě stisknutí tlačítka myši.



Obrázek 18: Wireframe pro hlavní obrazovku aplikace.



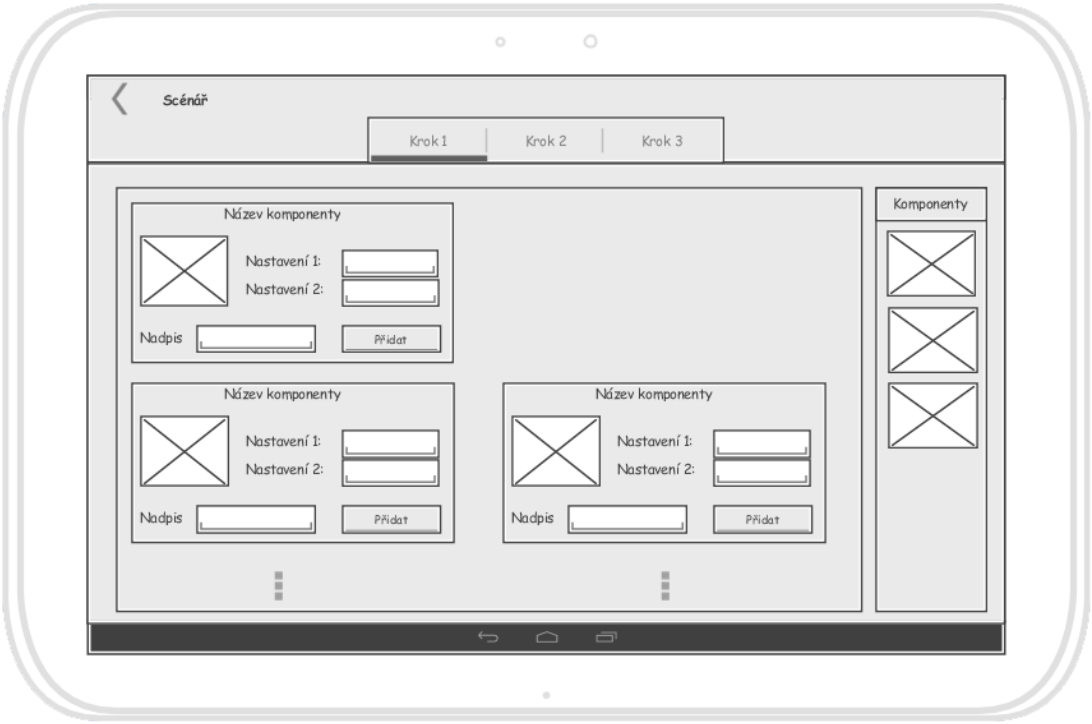
Obrázek 19: Wireframe pro hlavní obrazovku s menu navigací.

6.3.2 Tvorba scénářů

Vytváření scénářů se bude odehrávat na obrazovce, která bude přístupná skrz menu tlačítko s názvem "Scénář". Tato obrazovka bude členěná pomocí záložek (tzv. TabbedPage) a bude obsahovat tři záložky, které prezentují jednotlivé kroky pro vyváření scénářů. Navigaci obrazovek mezi jednotlivými kroky lze provést skrz kliknutí přímo na tlačítko daných kroků nebo vytvořením gesta na dotykové obrazovce z levé do pravé strany obrazovky nebo naopak.

V prvním kroku je zobrazena obrazovka s výběrem jednotlivých komponent pro scénář. Jednotlivé komponenty budou uskupeny podle použití a bude je možné přes tlačítko "přidat", přímo vložit do seznamu komponent. Tento seznam bude použit pro tvorbu a rozvržení scénáře a bude se nacházet na pravé části obrazovky. Součástí každé komponenty bude i její nastavení, které budou pro každou komponentu různé. Detailní popis nastavení komponent budou popsána s návrhem jednotlivých obrazovek.

Druhý krok představuje obrazovku pro tvorbu scénáře a jeho vzhledu. Scénář je možné skládat z několika oken, kterým lze nastavit rozložení obrazovky pro komponenty. V samotné levé častí obrazovky kroku 2 se bude nacházet seznam zobrazující přidané komponenty z předchozího kroku a v pravé části budou již přidané okna do scénáře. Mezi těmito seznamy se bude nacházet plocha pro vzhled okna. Do tohoto okna lze přenesením (tzv. Drag & Drop) ze seznamu komponent, přidat komponentu do okna scénáře. Oknu scénáře se následně nastaví název a doba zobrazení, po kterou se bude na hlavní obrazovce zobrazovat. Tlačítkem "přidat do scénáře"se přidá okno do scénáře a následně se zobrazí v seznamu oken na pravé straně, které představují jeden scénář. Ze seznamu komponent a oken scénáře bude také možnost pomocí Drag & Dropu na ikonku koše, smazat přidanou komponentu nebo vymazat celé okno ze seznamu.



Obrázek 20: Wireframe pro první krok tvorby scénářů.

Třetí krok je z velké části pouze zobrazovací. V pravé části stejně jako v předchozím kroku bude seznam jednotlivých oken daného scénáře. Po kliknutí na patřičné okno se uprostřed na obrazovce zobrazí okno scénáře, které vykreslí uživatelské prvky podle konfigurace jednotlivých komponent v daném okně. Součástí této obrazovky bude také ve spodní části možnost zadat název scénáře a pomocí tlačítka "uložit a uzavřít"se tento scénář přidá do seznamu již vytvořených scénářů a následně se uloží do souboru. Po vykonání těchto procesů se obrazovka s vytvářením scénáře zavře a zobrazí se hlavní obrazovka na které budou ve smyčce zobrazovány jednotlivé scénáře a jejich okna.

6.3.3 Aplikační nastavení

Pro zobrazení aplikačního nastavení musí uživatel stisknout tlačítko menu na hlavní obrazovce a vybrat volbu aplikační nastavení. Po stisknutí této volby bude následně uživatel přesměrován na obrazovku s přihlášením a budou-li přihlašovací údaje validní, bude ihned přesměrován na aplikační nastavení. Zde je obrazovka rozdělena na dvě části. Na levé části se nachází nastavení pro jednotlivé komponenty jako je velikost písma, velikost řádku a intervalová doba kalendáře, aktualizace dat se serverem nebo možnost vybrat téma aplikace, barvu textu a rozlišovací barvu aplikace. Na pravé straně se budou nacházet tlačítka pro import a export scénářů do souboru ve formátu JSON. Pod těmito tlačítky budou zobrazeny všechny vytvořené scénáře, které lze

pomocí tlačítek smazat nebo upravit. Dále je také možné jednotlivé scénáře zapnout nebo vypnout. Na základě tohoto nastavení se budou jednotlivé scénáře zobrazovat nebo nezobrazovat na hlavní stránce.

Po dokončení všech úprav na obrazovce musí uživatel aplikovat změny pomocí stisknutí tlačítka ""uložit". V případě, že by se uživatel vrátil na předchozí obrazovku přes tlačítko zpět, změny nebudou aktivovány. Součástí je také možnost, pomocí tlačítka "reset", resetovat nastavení a tím vrátit všechny nastavení do výchozích hodnot.

Obrázek 21: Wireframe pro aplikační nastavení aplikace.

6.3.4 Komponenty pro scénáře

Komponenty představují jednu z nejdůležitějších částí aplikace. Jedná se o znázornění různých typů informací do okna, které může být vloženo do scénáře a následně potom zobrazeno samostatně nebo s ostatními komponentami jako celek na hlavní obrazovce aplikace. Komponenty budou implementovány pouze jako informativní prvky a proto nebudou obsahovat žádné akce, které by mohl uživatel vyvolat. V aplikaci budou implementovány následující komponenty:

- **Probíhající rezervace**
  Jedná se o zobrazení právě probíhajících rezervací na daném zdroji. Zdroj může nabývat těchto tří typů: služba, zaměstnanec nebo místo. Hlavními zobrazovacími prvky na tomto okně jsou informace, které se budou řadit do prostřední části obrazovky. Jedná se o název

daného zdroje pod kterým budou informace s časem začátku a konce rezervace a jména jednotlivých zákazníků, kteří mají danou službu zarezervovanou. Součástí bude také počet rezervovaných zákazníku v pravém dolním rohu.

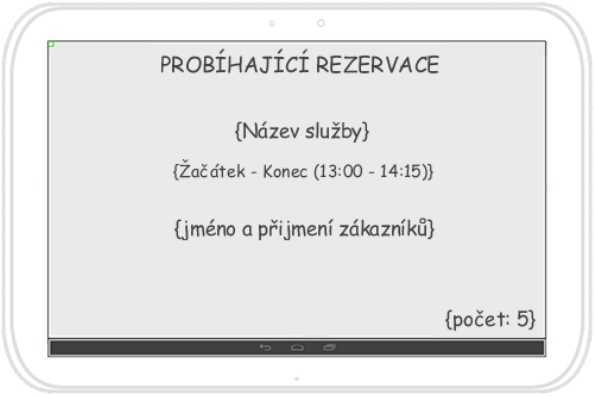
Uživatelské nastavení komponent

- Uživatel bude mít možnost vybrat pouze část zdrojů, pro které chce vidět právě probíhající rezervace.
- Současně bude mít také možnost zvolit počet vteřin po kterou bude právě probíhající rezervace na určitém zdroji zobrazena než se zobrazí rezervace z dalšího vybraného zdroje.

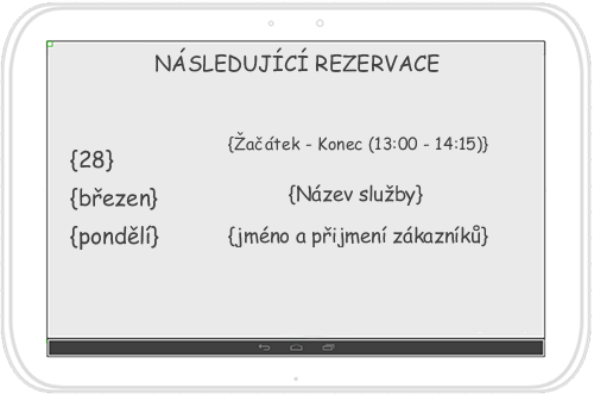
- **Následující rezervace**
  Komponenta pro následující rezervace bude zobrazovat rezervace, které ještě nezačaly, ale jsou v pořadí po právě probíhajících rezervací. Stejně jako v komponentě "probíhající rezervace"umožňuje zobrazení všech typů zdrojů. Obrazovka této komponenty bude zobrazovat na levé straně datum ve formátu den, měsíc a název dne. Na pravé části obrazovky budou zobrazeny informace pod sebou v následujícím pořadí, čas začátku a konce služby, název služby a následně jména všech rezervovaných účastníků.

  Uživatelské nastavení komponenty

  - V komponentě bude možné zvolit počet vteřin pro zobrazení jedné rezervace, než se zobrazí další. Stejný princip jako u komponenty "probíhající rezervace", kdy se jednotlivé rezervace střídají v zadaném časovém intervalu.
  - Bude zde možné zvolit také maximální počet rezervací. Aplikace pak bude zobrazovat na základě tohoto počtu, pouze počet nejbližších rezervací na všech zdrojích.



Obrázek 22: Wireframe pro komponentu: Probíhající rezervace.



Obrázek 23: Wireframe pro komponentu: Následující rezervace.

- **Přehled následujících rezervací**
  Jak už název komponenty napovídá, obrazovka bude zobrazovat přehled nejbližších rezervací na všech zdrojích podle zvoleného počtu. Rezervace budou řazeny do řádků, kdy zleva bude zobrazen čas začátku a konce rezervace, následovat bude její název. Jména jednotlivých registrovaných zákazníků k dané rezervaci budou v novém řádku a začínat budou pod názvem rezervace.

  Uživatelské nastavení komponenty

  – Pro tuto komponentu bude možné pouze zvolit počet maximálně zobrazovaných následujících rezervací na obrazovce.

- **Text a popis**
  Pro zobrazení libovolného textu v aplikaci je zapotřebí komponenta, která toto bude moci umožňovat. Jedná se o jednoduchou obrazovku zobrazující prostý text, který bude směřován ke středu obrazovky. Pro větší rozsah možností, zde bude možné zadat i druhý menší text, taktéž uprostřed. Oba texty budou nepovinné a jejich obsah bude záležet na uživateli.

  Uživatelské nastavení komponenty

  – Možnost zadat primární text (nadpis).

  – Možnost zadat sekundární text (popis).

- **Datum a čas** Součástí požadavků je zobrazení aplikace v tzn. Fullscreen módu. Z tohoto důvodu nebude možné vidět aplikační bar s datem a časem. Je proto potřeba vytvořit komponentu, která tuto funkci nahraní. Bude se jednat o klasické zobrazení data a času. Uprostřed na obrazovce bude v řádcích aktuální čas, následovat bude datum a také název aktuálního dne. Různé možnosti zobrazení bude možné volit v nastavení komponenty při její vytváření.

  Uživatelské nastavení komponenty

  – Na obrazovce bude možné nastavit k vidění aktuální čas se sekundami nebo bez.

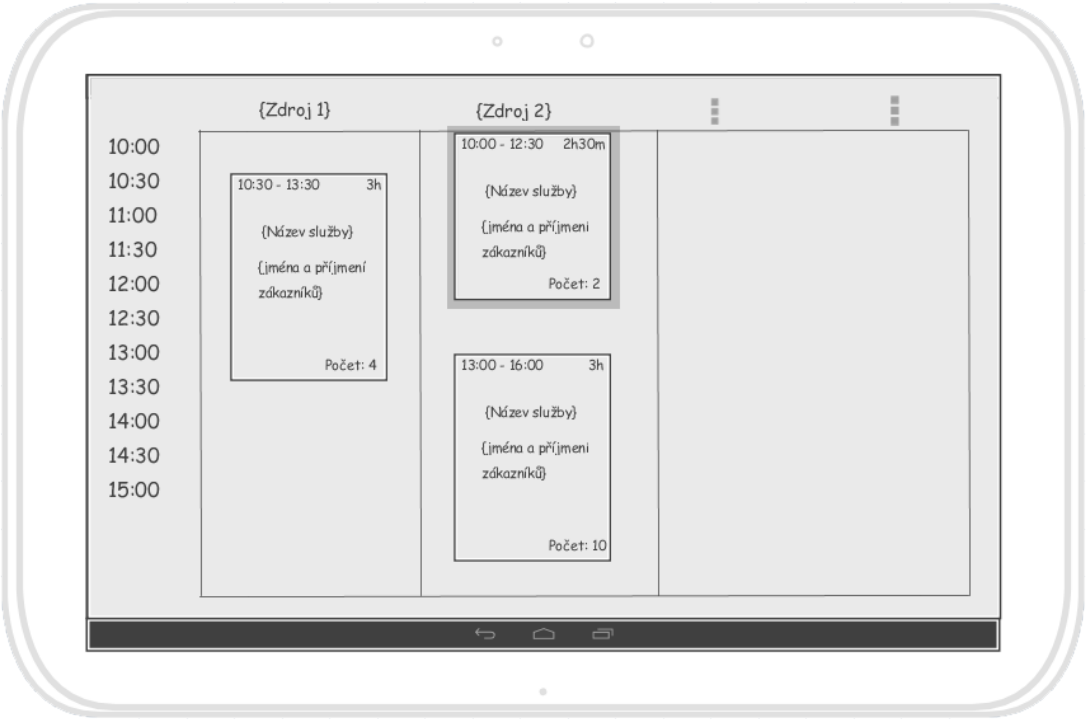
  – Možnost zobrazení se bude také vztahovat k datu a názvu aktuálního dne.

- **Celodenní přehled**
  Tato komponenta bude grafickým znázornění daného zdroje pro celý den. Obrazovka bude rozdělena podle časových úseků, kdy velikost těchto úseků lze nastavit v aplikačním nastavení aplikace. Následně se v těchto časových úsecích budou zobrazovat služby poskytované daným zdrojem, na základě její doby trvání. V jednotlivých službách je potřeba zobrazovat čas začátku a konce, dobu trvání, název služby a jména zákazníků, kteří jsou přihlášeni k dané službě. Je také potřeba nezapomenout, že jednotlivé služby se můžou v některých případech překrývat. Pro přehledné rozložení informací bude na levé straně obrazovky časová posloupnost a v horní části názvy jednotlivých zdrojů. Zbývající části obrazovky budou

využita právě k zobrazení jednotlivých služeb pro konkrétní zdroj. Současně také není zapotřebí, aby na obrazovce byly vidět staré rezervace. Pro tento účel plně postačí, když budou zobrazeny k aktuálnímu času rezervace, které byly v uplynulých dvou hodinách.

Uživatelské nastavení komponenty

– Jediným uživatelským nastavením pro tuto komponentu bude výběr možných zdrojů zobrazovaných na obrazovce s jejich službami.



Obrázek 24: Wireframe pro komponentu: Celodenní přehled.

- **Video**

  Pro požadavek na podporu multimediální obsahu aplikace, vznikla možnost spouštět v aplikaci videa. Proto je potřeba komponenta pro zobrazování a přehrávání videí, která bude zobrazovat video přes celou obrazovku aplikace. Zdroj videa bude řešen pomocí url odkazu k serveru (např. k Youtube).

  Uživatelské nastavení komponenty

  – Možnost vložit url odkaz do textového pole.

- **Přehled obrázků**
  K přehrávání videa patří i prezentace obrázků. Tudíž bude další komponenta představovat jednoduchý přehled obrázků. Obrázky budou zobrazeny přes celou obrazovku aplikace, podobně jako je tomu u komponenty videa. Stejně tak budou obrázky zobrazeny na základě url odkazu.

  Uživatelské nastavení komponenty

  - URL odkazy bude možné vkládat do oddělených textových polí (maximálně však 10 odkazů).
  - Možnost zadat časový interval na jehož základě se budou jednotlivé obrázky střídat.

- **Zpracováno rezervací**
  Prvním sumarizačním prvkem pro aplikaci je komponenta, která bude mít na starosti zobrazování součtu již zpracovaných rezervací. Jedná se o všechny rezervace, které již proběhly, vyjma těch, které jsou právě v probíhajícím stavu. Jednotlivé informace budou rozmístěny na obrazovce následujícím způsobem. Jestliže bude komponenta propojena s konkrétním zdrojem, bude nahoře uprostřed obrazovky název daného zdroje, v opačném případě při výběru všech zdrojů, bude toto pole prázdné. Následně bude uprostřed obrazovky zobrazena ikona pro rozpoznání, že se jedná o komponentu pro zpracované rezervace, pod kterou bude zobrazeno součet udávající počet zpracovaných rezervací.

  Uživatelské nastavení komponenty

  - Možnost volby zdroje (jeden konkrétní nebo všechny).

- **Rezervace ke zpracování** Další sumarizačním prvkem je komponenta s názvem "rezervace ke zpracování". Tato komponenta je velmi podobná komponentě "zpracováno rezervací"s rozdílem, že její hlavní princip není zpracovávat již proběhlé rezervace, ale zpracovávat ještě neproběhnuté včetně probíhajících. Rozlišovacím nástrojem této komponenty a komponenty "zpracováno rezervací"jsou právě ikony, které se nachází uprostřed daných obrazovek.

  Uživatelské nastavení komponenty

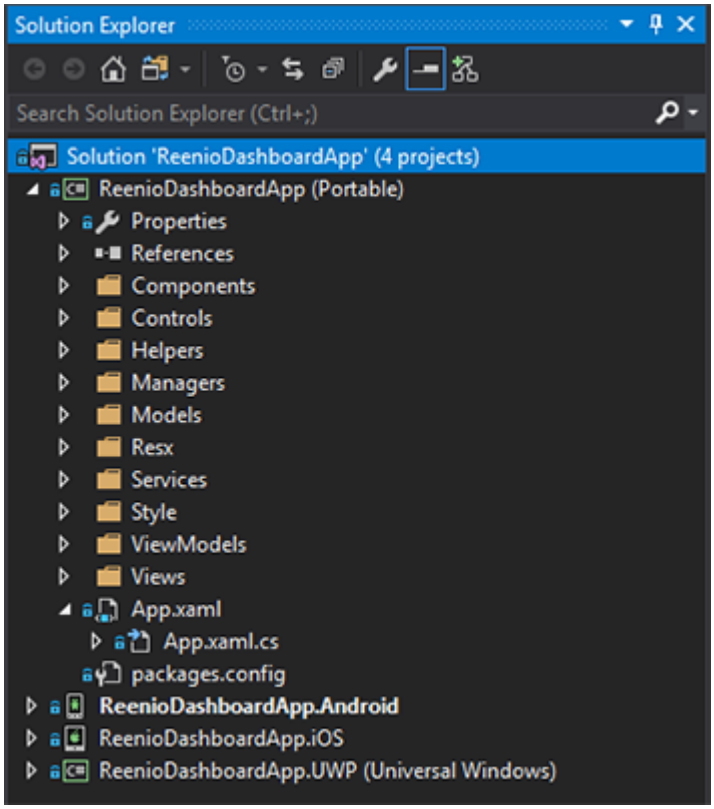
  - Možnost volby zdroje (jeden konkrétní nebo všechny)

7 Implementace

Součástí této práce byla také samotná implementace navržené multiplatformní aplikace pomocí zvoleného nástroje v prostředí Xamarin. Při volbě technologie byly uplatněny zjištěné vlastnosti ze studie prostředí Xamarin a jeho blízkého okolí. Z důvodu značné rozsáhlosti kódu aplikace, budou v této kapitole pouze naznačeny techniky, které jsem využil při implementaci. Jedná se především o ukázky kódu multiplatformního a nativního vývoje, které jsou v aplikaci použity. Součástí této kapitoly bude také ukázána reálná implementace komunikace s API rozhraní, která navazuje na práci s textovým řetězcem JSON.

7.1 Multiplatformní prvky

Hlavní a největší část implementované aplikace se nachází v knihovně PCL, která je sdílená pro projekty Android, iOS a UWP. Tato knihovna obsahuje všechny multiplatformní prvky, které jsem implementoval. Jedná se především o celé uživatelské rozhraní, business logiku, datový model a také komunikaci s API rozhraním. V následující části bude popsána struktura této knihovny podle jednotlivých složek.



Obrázek 25: Struktura projektu multiplatformní aplikace ve Visual Studiu 2017.

7.1.1 Start multiplatformní aplikace

Spuštění aplikace má na starost třída App, která je volána po nativní inicializaci při startu aplikace na dané platformě. V této třídě se definuje prvotní zobrazovací obrazovka a probíhá zde na pozadí komunikace s API rozhraním.

7.1.2 Model-View-ViewModel

Do stejnojmenných názvů složek je rozdělena funkcionalita aplikace podle návrhového vzoru MVVM. Ve složce Models se nachází třídy, které jsou současně jak pro datový model aplikace, tak i struktury pro přenos dat mezi aplikací a API rozhraním. Jedná se o třídy, které jsou přizpůsobené pro práci s JSON formátem (serializace a de-serializace) AccessDTO, ReservationsDTO, EmployeeDTO, PlaceDTO a ServiceDTO. Dále je zde vytvořených několik tříd, které slouží jako struktura pro ViewModely aplikace. Jsou to především třídy Scenario a ScenarioList.

```
public partial class AccessDTO
{
   [JsonProperty("accessToken")]
   public string AccessToken { get; set; }

   [JsonProperty("accessTokenPurpose")]
   public string AccessTokenPurpose { get; set; }

   [JsonProperty("user")]
   public User User { get; set; }

   [JsonProperty("subjects")]
   public Subject[] Subjects { get; set; }

   [JsonProperty("defaultSubjectId")]
   public long DefaultSubjectId { get; set; }
}
```

Výpis 4: Třída AccessDTO pro práci s JSON formátem.

Ve složce ViewModels se nachází třídy, ve kterých je implementovaná logika k daným View. Tyto třídy dále pracují s modely, na jejich základě se určují pravidla jak má View vypadat. Charakteristickým znakem těchto tříd je, že dědí z třídy BaseViewModel ve kterém se nachází společná implementace, jako je např. událost pro změnu Property a metody, které se volají po otevření a zavření dané obrazovky. Jednotlivé ViewModely budou následně popsány:

- AppSettingViewModel - je třída, která představuje logickou část AppSettingView obrazovky. Je zde implementováno několik příkazů (Commands), které se starají např. o import a export scénářů, uložení a resetování aplikačního nastavení atd..
- LoginViewModel - třída, která řeší získání klíče z API rozhraní na základě přihlašovacích údajů.
- MainpageContentViewModel - obsahuje podstatnou část logiky celé aplikace. Tento ViewModel slouží pro hlavní obrazovku, která zobrazuje jednotlivé scénáře. Je zde implementovaná logika pro načítání všech aktivních scénářů, kterým se následně v cyklu asynchronní metody postupně vytvářejí instance View. A právě tyto instance jsou zobrazovány jako okna na hlavní obrazovce.
- SettingStep1ViewModel - ViewModel pro tvorbu scénářů v prvním kroku. Nachází se zde implementace, která má na starost vytvářet instance komponent. Jedná se o vytvoření modelu dané komponenty a přidání do seznamu pro použití v následujících krocích.
- SettingStep2ViewModel - v tomto kroku se se řeší vytváření jednotlivých oken pro daný scénář pomocí seznamu komponent, ze kterého se vybírají prvky a "Drag and Dropem"se přenáší do aktuálního vytvořeného okna. Následně se takových oken vytváří kompletní scénář.
- SettingStep3ViewModel - posledním ViewModelem je třída pro třetí krok, která implementuje logiku pro vytváření instancí jednotlivých oken scénářů a tím umožňuje uživateli vidět reálný vzhled scénáře před samotným uložením. Zároveň je zde implementován kód pro práci se soubory.

```
public class LoginViewModel : BaseViewModel
{
   private string _emailEntry = string.Empty;
   // code..
   public ICommand LoginCommand { get => new Command(Login); }

   public string EmailEntry
   {
      get => _emailEntry;
      set
      {
         _emailEntry = value;
         OnPropertyChanged(nameof(EmailEntry));
      }
   }
```

```
    private async void Login()
    {
        if (await LoginManager.Instance.CheckLoginPermission(EmailEntry,
            PassEntry))
        {
            SettingsManager.SetLogin(PermanentlySignedUp, EmailEntry, PassEntry);
            await NavigationManager.Instance.PushAsync(new AppSettingView());
        }
        // code...
    }
    // code..
}
```

Výpis 5: Ukázka části třídy LoginViewModel.

Poslední složkou pro MVVM vzor je složka Views. Tato složka obsahuje všechny třídy představující uživatelské rozhraní, které se nachází v aplikaci. Jednotlivé třídy jsou implementovány pomocí jazyku XAML a společně s jazykem C#, který představuje tzv. code behind neboli kód na pozadí, tvoří uživatelské rozhraní jedné obrazovky. Propojení aplikační logiky a uživatelského rozhraní se mapuje právě v code behind View, kde se vytvoří vazba mezi View a ViewModelem, tento proces se nazývá Binding Context.

Ve Views jsem implementoval několik odlišných druhů obrazovek. Jedná se např. o tyto typy:

- MasterDetailPage - tato obrazovka se skládá ze dvou rozdílných obrazovek, kde jedna je hlavní (Detail) a druhá znázorňuje menu aplikace, které vyjíždí z levé strany obrazovky (Master). Tímto typem je implementována hlavní obrazovka na které se zobrazují jednotlivé scénáře. Jedná se o třídu MainPageView, která v XAML kódu mapuje třídu MainPageMenuView (vyjíždějící menu) a třídu MainPageContentView (hlavní obrazovka). Na pozadí je implementována funkcionalita pro navigaci obrazovek z nabídky menu.

- TabbedPage - obrazovka, která umožňuje zobrazovat stránky vedle sebe jako záložky a přechod mezi nimi je řešen pomocí navigační lišty nebo vytvořením gesta z jedné strany obrazovky na druhou. V aplikaci je řešena tímto typem obrazovky, obrazovka pro vytváření scénářů s jednotlivými kroky. Hlavní TabbedPage obrazovka se nazývá SettingsView, která uvnitř XAML kódu implementuje ContentPage jednotlivých kroků. Jedná se o třídy SettingsStep1View, SettingsStep2View a SettingsStep3View.

- ContentPage - je jednoduchá obrazovka, která zobrazuje grafické komponenty přes celou obrazovku aplikace. Pomocí tohoto standardního druhu obrazovek jsou implementovány View pro aplikační nastavení (AppSettingView), přihlašovací obrazovka (LoginView) a obrazovka pro start aplikace (StartView).

```
<ContentPage ...>
   <ContentPage.Resources>
      <ResourceDictionary MergedWith="theme:DefaultStyle" />
   </ContentPage.Resources>

   <Grid>
      <StackLayout Grid.Row="0" Grid.Column="0" Padding="10"
          HorizontalOptions="FillAndExpand" VerticalOptions="FillAndExpand">
         <ContentView Content="{Binding ScenarioView}" HorizontalOptions="
             FillAndExpand" VerticalOptions="FillAndExpand" />
      </StackLayout>

      <StackLayout Grid.Row="0" Grid.Column="0">
         <StackLayout.GestureRecognizers>
            <TapGestureRecognizer Tapped="TapGestureRecognizer_Tapped"
                NumberOfTapsRequired="1" />
         </StackLayout.GestureRecognizers>
      </StackLayout>
   </Grid>
</ContentPage>
```

Výpis 6: Ukázka z XAML třídy MainPageContentView.

7.1.3 Manažeři a pomocné třídy

Ve složce Managers se nachází třídy, které vytváří specifickou část logiky aplikace, která nemůže být implementovaná ve ViewModelech, protože je využívaná více třídami. Jedná se např. o tyto třídy:

- RequestManager - statická třída, která implementuje veškerou funkcionalitu pro komunikaci s API rozhraním. Jde např. o získání tokenu, rezervací, zdrojů atd.. Všechny metody v této třídě jsou definovány jako statické a asynchronní a pro přístup k API využívají konstantní proměnné definované na začátku třídy, které definují url API cestu.
- LayoutViewManager - manažer, který vytváří grafické rozhraní pomocí jazyku C# na základě scénáře. Podle vstupních parametrů vytvoří obrazovku, která může být rozdělena do několika sloupců a řádků. Současně tato třída obsahuje také mechanismus, který vytváří možnosti pro vkládání jednotlivých komponent do daného vzhledu.

- MessageManager - soubor, který obsahuje několik tříd pro třídu MessagingCenter. Tyto třídy slouží jako struktura pro přenos dat. U klasických .NET událostí (Event) by se jednalo o třídy podobné třídě EventArgs.
- SettingsManager - třída, která má za úkol řešit veškeré operace týkající se aplikačního nastavení. Je zde implementováno několik příkazů (Command), která načítají, ukládají nebo resetují nastavení aplikace. Mimo jiné je zde implementována také událost (Event), která informuje své posluchače o změně velikosti písma pro všechny použité komponenty.
- ViewManager - statická třída, která obsahuje logiku pro vytváření instance View jednotlivých komponent na základě jejich modelu. Pro velkou úsporu a přehlednost kódu je využita dynamičnost pomocí rozhraní IComponent, která implementuje každý model komponenty. Na základě tohoto rozhraní se uvnitř třídy přetypuje vložený parametr a podle správné třídy se vytvoří instance, které je následně metodou navrácena.
- JsonManager - důležitý manager, který má na starosti práci se scénáři. Jedná se hlavně o správné načtení JSON souboru z interního úložiště a jeho následnou de-serializaci do třídy pro scénáře (ScenarioList). V opačném směru také o serializaci dat do formátu JSON a jeho následné uložení do souboru na interní úložiště.

Složka Helpers obsahuje pomocné třídy aplikace, které nejsou úrovňově tak důležité jako třídy nacházející se ve složce Managers. V této složce se nachází následující třídy:

- LogHelper - třída, která implementuje funkcionalitu pro vytváření log souboru. V praxi se jedná o zápis chyby do souboru, která je odchycena promocí tzv. try-catch bloku. Tato třída je implementována pomocí vzoru Singleton, aby bylo zajištěno vytvoření pouze jedné její instance.
- TimerHelper - obsahuje implementaci pro vytvoření časovače. Jelikož proti .NET frameworku, kde stačí importovat baliček s názvem Timer a použít již vytvořené třídy, v Xamarin.Forms zatím není tato funkcionalita přístupná. Z tohoto důvodu je v aplikaci implementovaný vlastní časovač, který lze zapnout a vypnout pomocí metod Start a Stop.
- TimerLoadHelper - třída, která definuje aktuální datum a čas pro aplikaci. Součástí třídy je také definice pro začátek a konec dne ve kterém aplikace načítá a zobrazuje data.

7.1.4 Komponenty scénáře

Ve složce Components se nachází všechny implementované komponenty, které aplikace obsahuje. Každá komponenta představuje jeden soubor a je implementovaná pomocí MVVM vzorce. View je reprezentováno pomocí jazyka XAML, ViewModel a Model se nachází v code behind v samostatných oddělených třídách. Komponenty jsou implementovány tak, aby bylo možné je využít i v úplně jiných projektech, než jenom v těch, které se nachází v současné aplikaci.

Je zde implementovaných všech 8 komponent, které byly navrženy a popsány v dřívější kapitole zabývající se samotným návrhem aplikace.

7.1.5 Vlastní grafické prvky

Složka Controls obsahuje několik tříd, které jsou implementované v XAMLu nebo v C#. Každá třída představuje vlastní vytvořený grafický prvek, který se dá použít v uživatelském rozhraní aplikace. Vlastní grafické prvky jsem vytvořil, z důvodů minimalizace a zamezení duplicity kódu na některých místech aplikace. Pro představu se zde nacházejí např. tyto implementované grafické prvky:

- ComponentsDraggedListView - prvek, který představuje okno pro "Drag & Drop"itemů ze seznamu do seznamu. Tento prvek se převážné využívá při vytváření scénářů a jejich vzhledu.
- PopUpPage - uživatelský prvek, který nahrazuje vyskakující okno (Pop Up), které je typické na mobilních platformách. Implementace je provedena pomocí jazyka C#. Tento prvek překrývá obrazovku průhlednou černou barvou, která zobrazuje uprostřed potřebné informace na základě aktuální obrazovky.
- LoadingIndicator - jednoduchá obrazovka, která překrývá aktuální obrazovku. Používá se převážně při čekání na dokončení nějaké akce. Jedná se o typickou obrazovku s točícím indikátorem "waiting".

7.1.6 Služby

Složka Services, která obsahuje rozhraní pro Dependency Service. Nachází se zde rozhraní ICloseApplication s jedinou metodou CloseApp pro předpis tříd, které jsou vytvořeny na každé platformě a mají za úkol implementovat tuto metodu s logikou pro ukončení aplikace. Dalším rozhraním je ISaveAndClose, která slouží pro načítání a ukládání textu do souboru.

Současně se v této složce nachází třídy, které mají na starosti při spuštění aplikace, vytvářet požadavky na API rozhraní a dále je zpracovávat. Jsou zde tyto třídy:

- ReservationManager - singleton třída, která při vytvoření instance spustí cyklus pro stahování všech rezervací ze systému pomocí specifikovaného času. Tento cyklus posílá požadavky na základě doby, která je stanovená v aplikačním nastavení. Po dokončení stažení projdou všechny rezervace algoritmem, který z nich vytvoří úkoly pro dané zdroje. Následně jsou úkoly a rezervace uloženy do veřejných proměnných, které jsou dále využívány jednotlivými komponenty.
- ResourcesManager - třída, která má za úkol v intervalu stahovat zdroje z rezervačního systému a dále je poskytovat pro aplikaci.

7.1.7 Ostatní třídy

Za zmínku stojí např. třídy pro výstupní text na obrazovce aplikace (Resources). Pomocí těchto tříd se řeší více-jazyčnost aplikace. Implementovaná aplikace podporuje výchozí anglický jazyk, který je definován ve třídě AppResources.resx a v třídě AppResources.cs.resx je reprezentován český jazyk.

Důležitou součástí je také třída pro styl grafických prvků aplikace. Tato třída se nazývá DefaultStyle.xaml a implementuje styly pro všechny použité grafické prvky v aplikaci, jsou to např. label, text block, grid, stack layout a mnoho dalších.

7.2 Nativní prvky

Při implementaci aplikace pomocí technologie Xamarin.Forms jsem narazil na problém chybějících funkcionalit nebo možnost jejich vlastních úprav. Ve frameworku existuje možnost, kterou je tato problematika chybějící funkcionalit řešena pomocí tzv. Dependency Service a umožňuje implementovat a využívat nativní vlastnosti dané platformy. Stejně tak i úprava uživatelských prvků a její logiky je v Xamarin.Forms umožněna. Tento proces je vyřešen pomocí vytvoření nových prvků za pomocí dědění a nazývá se Custom Renderer.

7.2.1 Custom Renderer

Custom Renderer je princip, který umožňuje vývojáři upravit chování nebo vzhled jakýchkoli ovládacích prvků a to zvlášť pro každou platformu. Princip spočívá v definování vlastní třídy, která je potomkem daného prvku. Tato třída obsahuje většinou Bindable Property a nacházet se ve sdílené knihovně aplikace. Následně jsou na každé platformě vytvořeny nové třídy (tzv. renderery), které jsou odkazovány na již vytvořenou třídu ve sdílené knihovně. V těchto rendererech se následně nachází veškerá nová logika prvku, která je do implementována na základě vytvořených Bindable Property.

V aplikaci, kterou jsem implementoval několikrát nastaly potřeby modifikovat samotné uživatelské prvky, protože neobsahovaly požadovanou funkcionalitu. Jeden z těchto problému vznikl při nutnosti překrýt obrázek specifickou barvu, kterou bude možné kdykoli měnit. Na následující ukázce je zobrazen renderer, který jsem implementovat pro vyřešení tohoto problému.

Následující kód zobrazuje třídu, které je implementovaná ve sdílené knihovně (PCL). Jedná se o vytvoření vlastního ovládacího prvku na základě již existujícího View. V tomto případě jsem přidal možnost definovat barvu a cestu ke zdroji, kde se nachází obrázek. Obě nové vlastnosti jsou definovány jako Bindable Property, aby bylo možné jejich zadání v XAML kódu.

```
public class IconView : View
{
   public static readonly BindableProperty ForegroundProperty =
       BindableProperty.Create(nameof(Foreground), typeof(Color), typeof(
       IconView), default(Color));

   public static readonly BindableProperty SourceProperty = BindableProperty.
       Create(nameof(Source), typeof(string), typeof(IconView),default(string));

   public Color Foreground
   {
      get => (Color)GetValue(ForegroundProperty);
      set => SetValue(ForegroundProperty, value);
   }

   public string Source
   {
      get => (string)GetValue(SourceProperty);
      set => SetValue(SourceProperty, value);
   }
}
```

Výpis 7: Ukázka třídy IconView v knihovně pro sdílený kód.

Samotná třída vlastního rendereru pro platformu Android se nachází přímo v projektu Xamarin.Android. Pomocí atributu ExportRendererAttribute, který je podmíněn dvěma parametry – typ vlastního prvku a typ rendereru, následně Xamarin pozná, že má jít o vlastní vykreslení prvku. Na následující ukázce Android rendereru, lze vidět na atributu propojení s třídou IconView. Při použití této třídy se následně vyhledá instance rendereru, kde budou použity její přepsané metody.

```
[assembly: ExportRendererAttribute(typeof(IconView), typeof(IconViewRenderer))]
namespace BoardReservationsApp.Droid.Renderers
{
   public class IconViewRenderer : ViewRenderer<IconView, ImageView>
   {
      // code..
   }
}
```

Výpis 8: IconViewRenderer v Android projektu.

Hlavní metodou tohoto rendereru je metoda SetImage, která má na starost vykreslování obrázku na plátno dané platformy a následně ho přebarví zvolenou barvou.

```
private void SetImage(IconView previous = null)
{
   if (previous == null)
   {
      var uiImage = new UIImage(Element.Source);
      uiImage = uiImage.ImageWithRenderingMode(UIImageRenderingMode.
          AlwaysTemplate);
      Control.TintColor = Element.Foreground.ToUIColor();
      Control.Image = uiImage;

      if (!_isDisposed)
         ((IVisualElementController)Element).NativeSizeChanged();
   }
}
```

Výpis 9: Ukázka metody SetImage v iOS rendereru.

Vytvořený nový uživatelský prvek je možné použít přímo ve View pomocí standardního přístupu. Na následujícím kódu lze vidět vytvoření obrázku a překrytí aplikační barvou, která je definována pomocí Bindingu.

```
<controls:IconView Grid.Column="0" Grid.Row="0" Grid.RowSpan="2" Source="
    AllDayIcon" Foreground="{DynamicResource MyColor}" Style="{StaticResource
    Step1ComponentImage}" />
```

Výpis 10: Použití prvku IconView v XAMLu.

V aplikaci jsem implementoval ještě další renderery, přičemž za zmínku stojí hlavně renderer pro Drag & Drop funkci, která umožňuje přesouvat prvky ze seznamu do jiného seznamu, dokonce i do seznamů rozdílných datových typů. Bohužel jsou tyto renderery velmi rozsáhlé a nebylo by vhodné je prezentovat jako ukázku rendererů v této práci. Jedná se především o tyto renderery:

- DragAndDropListViewRender
- DragAndDropCellRenderer

7.2.2 Dependency Service

Chybějící funkcionalita v multiplatformním kódu je řešena pomocí tzv. Dependency Service. Tato funkcionalita je součástí frameworku Xamarin.Forms. Jedná se o systém, který umožňuje sdílené knihovně volat specifické metody a funkce, které jsou implementovány pouze pro nativní platformu.

Za pomocí předloženého rozhraní dokáže Dependency Service najít správnou implementaci v dané platformě a spustit ji. Pro správnou funkčnost tohoto systému je zapotřebí definovat interface ve kterém je specifikovaná daná funkcionalita. Dále je potřeba v každém nativním projektu vytvořit implementaci s danou interface a následně tuto implementaci zaregistrovat, aby ji Dependency Service mohla najít.

V aplikaci se naskytly dva problémy, které nebylo možné řešit pomocí standardních funkcí v Xamarin.Forms. Jedná se o ukončení aplikace a práci se soubory, jako je ukládání a načítání souboru z interní nebo externí paměti.

Na následující ukázce je vidět implementace pro ukončení aplikace. Jedná se o vytvoření třídy přímo v nativním projektu. Pomocí atribut Dependency s parametrem typu dané třídy a rozšíření třídy o určitý interface, dokáže framework Xamarin.Forms najít a vytvořit instanci této třídy.

```
[assembly: Dependency(typeof(CloseApplication_Android))]
namespace BoardReservationsdApp.Droid.Services
{
   public class CloseApplication_Android : ICloseApplication
   {
      public void CloseApp()
      {
         Android.OS.Process.KillProcess(Android.OS.Process.MyPid());
      }
   }
}
```

Výpis 11: Dependency service pro ukončení Android aplikace.

Použití následující implementace ve sdílené knihovně, lze vytvořit instanci nativní funkcionality a dále sní pracovat.

```
var closer = DependencyService.Get<ICloseApplication>();
if (closer != null)
   closer.CloseApp();
```

Výpis 12: Ukázka z XAML třídy.

Jelikož Xamarin.Forms nedokáže pracovat se soubory, bylo nutné implementovat i tuto funkcionalitu, pomocí Dependency Service. Tato funkcionalita bude popsáná následující kapitole, která se zabývá práci s daty.

7.3 Práce s daty

7.3.1 API

Pro získávání dat z databáze rezervačního systému poslouží tzv. prostředník. Práci prostředníka mezi aplikací a touto databází bude mít na starosti API webové rozhraní. V aplikaci jsem pro tuto komunikaci mezi aplikací a API rozhraním implementovat pomocí funkcionality pro odesílání HTTP požadavku (GET, POST), který je odesílán na API rozhraní, kde je zpracován a následně je odeslána odpověď na tento dotaz zpět ve formátu JSON.

Na základě požadavků pro rezervační systém, jsem implementoval univerzální metody pro komunikaci s API rozhraním systému, který je možné specifikovat pomocí proměnné definující url odkaz k danému systému. Pomocí třídy HttpClient, která je součástí .NET frameworku, jsem implementoval následující metody:

- GetAccessToken - zajišťuje získání unikátního klíče (tokenu) a subjekt id (unikátní číslo uživatele), na základě uživatelských přihlašovacích údajů, které jsou pomocí parametrů předány metodě. Získání validního klíče ze systému je podmínkou pro odesílání validních dat z API rozhraní, které jsou požadované z ostatních požadavků aplikace (např. získání rezervací na určitý den).

  Metoda GetAccessToken je implementována jako HTTP POST požadavek, který je vhodný zejména pro jeho vlastnost nemodifikovat samotné URL specifickými daty, ale odesílat tyto data jako součást HTTP dotazu. Pro zamezení viditelnosti dat jako je posílané heslo a uživatelské jméno je využit právě požadavek POST.

- GetReservations - metoda vrací pro daného uživatele list rezervací, které se nachází v určitém časovém rozmezí, předávané pomocí parametrů "od-do"s datovým typem Datetime.

  Tato metoda představuje HTTP požadavek GET, který modifikuje url odkaz, přidáním unikátního klíče a subjekt id.

- GetResources - jedná se o metodu, která přijímá parametr datového typu pro rozlišení zdroje, které má načítat. Zdroje mohou představovat zaměstnance, místa a nebo služby. Po provedení požadavku jsou vráceny všechny uživatelovy vytvořené zdroje pro daný typ.

  Metoda je implementována na stejném principu jako metoda GetReservations. Jedná se tedy o HTTP požadavek typu GET, který vzhledem k metodě GetReservations ještě přidává a modifikuje url adresu názvem typu zdroje.

Následující kód představuje část implementace metody GetReservations, která pomocí klíče a subjekt id odešle dotaz pro rezervace v určitém intervalu. Následně doručenou odpověď v

JSONu zpracuje a překonvertuje do třídy určené pro rezervace (ReservationDTOCollection), která představuje list rezervačních tříd (ReservationDTO).

```
public static async Task<ReservationDTOCollection> GetReservations(DateTime
    start, DateTime end)
{
   var queryParams = new Dictionary<string, string>();
   queryParams.Add("start", ConvertDateTime.ToString(start));
   queryParams.Add("end", ConvertDateTime.ToString(end));

   using (HttpClient client = new HttpClient())
   {
      // code..
      var url = _reservationsUrl + "?" + string.Join("&", queryParams.Select(x
          => x.Key + "=" + x.Value));
      var result = await client.GetAsync(url);
      var response = await result.Content.ReadAsStringAsync();

      collection = ReservationDTOCollection.FromJson(response);
   }
   // code..
}
```

Výpis 13: Implementace části metody GetReservations.

Důležitá součást těchto metod, kterou je potřeba zmínit, je implementace asynchronního zpracování logiky uvnitř metod. Použitím modifikátorů async a await je možné zavolat metodu na jiném vlákně. Tímto dokážeme zajistit, aby aplikace nepřestala reagovat na požadavky uživatele, zatím co provádí nějakou dlouhotrvající operaci, jako můžou být právě požadavky na API rozhraní a jejich následné zpracování.

7.3.2 Internal storage

Pro práci s jednotlivými scénáři (jako je načítání, importování, exportování atd.) je potřeba využívat úložiště zařízení, na kterém běží implementovaná aplikace. U zařízení můžeme rozlišovat interní a externí úložiště, jelikož externí úložiště nemusí být vždy dostupné, je pro aplikaci využíváno vždy právě interní úložiště. Z důvodu chybějící funkcionality ve frameworku Xamarin.Forms, který neobsahuje práci se soubory pro podporované platformy, bylo nutné implementovat tuto funkcionalitu skrze nativní metody pomocí Dependency Service.

Na následujícím kódu jsou vidět implementované metody pro platformu Android. Jedná se o metody pro uložení textu (SaveTextAsync), pro načtení textu (LoadTextAsync) a metoda pro vytvoření cesty k veřejné složce (CreatePathToFile).

```
[assembly: Dependency(typeof(SaveAndLoad_Android))]
namespace BoardReservationsApp.Droid.Services
{
   public class SaveAndLoad_Android : ISaveAndLoad
   {
      public string GetPathOfFile(string filename)
      {
         var documentsPath = global::Android.OS.Environment.
             GetExternalStoragePublicDirectory(global::Android.OS.Environment.
             DirectoryNotifications);

         return Path.Combine(documentsPath.AbsolutePath, filename);
      }
      // code..
   }
}
```

Výpis 14: Ukázka implementace třídy pro práci se soubory na platformě Android.

Metody pro načítání a ukládání textu ze souboru jsou totožné i v implementaci pro iOS, jediným rozdílem je přístup ke sdílené složce pro práci se soubory. V následující ukázce kódu lze vidět získání přístupu k této složce.

```
[assembly: Dependency(typeof(SaveAndLoad_iOS))]
namespace BoardReservationsApp.iOS.Services
{
   public class SaveAndLoad_iOS : ISaveAndLoad
   {
        public string GetPathOfFile(string filename)
      {
         var documentsDirUrl = NSFileManager.DefaultManager.GetUrls(
             NSSearchPathDirectory.DocumentDirectory, NSSearchPathDomain.User).
             Last();

         return string.Format(@"{0}\{1}", documentsDirUrl.Path, filename);
      }
   }
}
```

Výpis 15: Ukázka implementace získání cesty k souboru na platformě iOS.

7.3.3 Práce s JSON

Aplikace při práci se soubory využívá textový řetězec JSON, stejně tak v tomto formátu přijímá i odpověď odeslanou z API rozhraní na základě určitého požadavku. Pro práci s tímto textovým formát se standardně využívá .NET třída JsonConvert. Pomocí statické metody SerializeObject s listem objektů předaným jako parametr, lze vytvořit textový řetězec ve formátu JSON. V opačném případě pro získání listu objektů z textového řetězce, se používá statická metoda DeserializeObject<T>, které je zapotřebí definovat výstupní datový typ pro list objektů.

Na následujícím kódu je zobrazena ukázka implementace serializace listu pro scénáře do výstupního formátu JSON, který je následně uložen do interního úložiště zařízení.

```
public async static void SaveListScenarioAsJsonFile(List<Scenario> collection)
{
   try
   {
      var indented = Formatting.Indented;
      var settings = new JsonSerializerSettings();

      var json = JsonConvert.SerializeObject(collection, indented, settings);

      await FileManager.WriteTextAllAsync(_fileNameListOfScenarios, json);
   }
   catch(Exception ex)
   {
      LogHelper.Instance.LogException(ex.ToString(), nameof(JsonHelper));
   }
}
```

Výpis 16: Metoda pro uložení scénářů do souboru ve formátů JSON.

Následuje ukázka kódu pro načtení JSONu z interního úložiště a jeho de-serializace do instance listu scenarios.

```
public static async Task<List<Scenario>> LoadListScenarioFromJsonFile()
{
   try
   {
      var json = await FileManager.ReadAllTextAsync(_fileNameListOfScenarios);
      var settings = new JsonSerializerSettings();
```

```
            var scenarios = JsonConvert.DeserializeObject<List<Scenario>>(json,
                settings);

            return scenarios;
        }
        catch(Exception ex)
        {
            LogHelper.Instance.LogException(ex.ToString(), nameof(JsonHelper));

            return null;
        }
    }
```

Výpis 17: Metoda pro načtení a de-serializace scénářů ze souboru.

JSON formát pro třídu Scenario může vypadat následovně.

```
{"Scenario":
    {"Name": "Scenario 1",
     "IsActive": true},
     "Id": 1,
     "ComponentItems": [
     {"Id": "1", "Title": " Komponenta 1", "UseDefaultFont": "True", "Icon":
         SwitchNext},
     {"Id": "2", "Title": " Komponenta 2", "UseDefaultFont": "True", "Icon":
         ActualReservation},
     ]}
}
```

Výpis 18: Ukázka JSONu pro třídu Scenario.

7.4 Zhodnocení vývoje

Vývoj multiplatformní aplikace pomocí technologie Xamarin je výrazně rychlejší a z mého pohledu také jednodušší, než vývoj pomocí jiných multiplatformních nástrojů. Pomocí kombinace jazyků XAML a C#, lze vytvořit přehledný a čitelný kód, který je velmi přívětivý. Samotný vývoj aplikace nebyl úplně bez problému, jelikož s chybějící funkcionalitou, musely být doimplementovány některé potřebné funkce. Tyto implementace byly mnohdy zdlouhavé a náročné, protože jejich funkcionalitu bylo potřeba otestovat na více zařízení. Stávalo se totiž, že daná funkcionalita nefungovala správně na různých verzích systému. Tato chybovost je zapříčiněna přímo v jádru Xamarinu, ale vývojáři tohoto nástroje se je snaží neustále opravovat.

8 Android – pohled na výslednou aplikaci

Publikování výsledné Android aplikace je jeden z posledních kroků celého vývoje. Pro vystavení aplikace široké veřejnosti bude aplikace publikována pomocí internetového obchodu Google Play, ve kterém bude aplikace dostupná pro instalaci na uživatelské zařízení. V této kapitole bude popsán proces publikování Android aplikace do internetového obchodu Google Play.

8.1 Proces nasazení aplikace

V první řadě, jestliže chceme publikovat aplikaci do obchodu Google Play, je potřeba založit si u tohoto obchodu účet. Dále je nutné vytvořit certifikát, kterým se aplikace uzamkne proti neoprávněné publikaci. A následně zkompilovanou aplikaci s certifikátem je již možné publikovat na samotný Google Play pomocí vytvořeného účtu. Jednotlivé kroky těchto procesů jsou podrobněji popsány v příloze A.1.

8.2 Specifikace

V současné době je aplikace zkompilovaná a je možné ji spustit na Android zařízení ve verzi minimálně 4.4 s označením KitKat a maximální podporovaná verze je v aktuálním vydání Android 8.0 nazývaná také jako Android Oreo. Stáhnutí a následnou instalaci aplikace do zařízení je možné z obchodu Google Play pomocí následujícího odkazu. `https://play.google.com/store/apps/details?id=garvis.reenio.panel&hl=cs&ah=H0oTtakgHyl4YwRqTwGPYBXEV0U`

9 iOS – pohled na výslednou aplikaci

Aplikace z frameworku Xamarin.iOS může být distribuována pro operační systém iOS pouze v jednom jediném internetovém obchodu, který spravuje Apple a nazývá se App Store. Tato kapitola obsahuje průvodce krok za krokem, které jsou nutné k publikaci do tohoto internetového obchodu. Budou zde popsány především informace jak připravit aplikaci pro distribuci, jak používat nástroje společnosti Apple k podání žádosti o schválení aplikace a nakonec i její samotné publikování do App Storu.

9.1 Proces nasazení aplikace

Příprava pro publikaci iOS aplikace je mírně náročnější než pro platformu Android. V první řadě procesu je potřeba založit účet pro obchod Apple Store. Druhým krokem je vytvoření certifikátu a konfigurace distribučního profilu pro zabezpečení aplikace. Následně je potřeba zkompilovat aplikaci do IPA souboru, nakonfigurovat iTunes Connect a aplikace se může nahrát do App Store. Celý proces tohoto nasazení je popsán v přehledných krocích v příloze A.2.

9.2 Specifikace

V současné době není vytvořen aktuální release verze implementované aplikace pro systém iOS a tudíž není ani aplikace přístupná na App Store. Důvodem chybějícího instalačního balíčku je nedostupnosti Mac zařízení s operačním systémem OS.

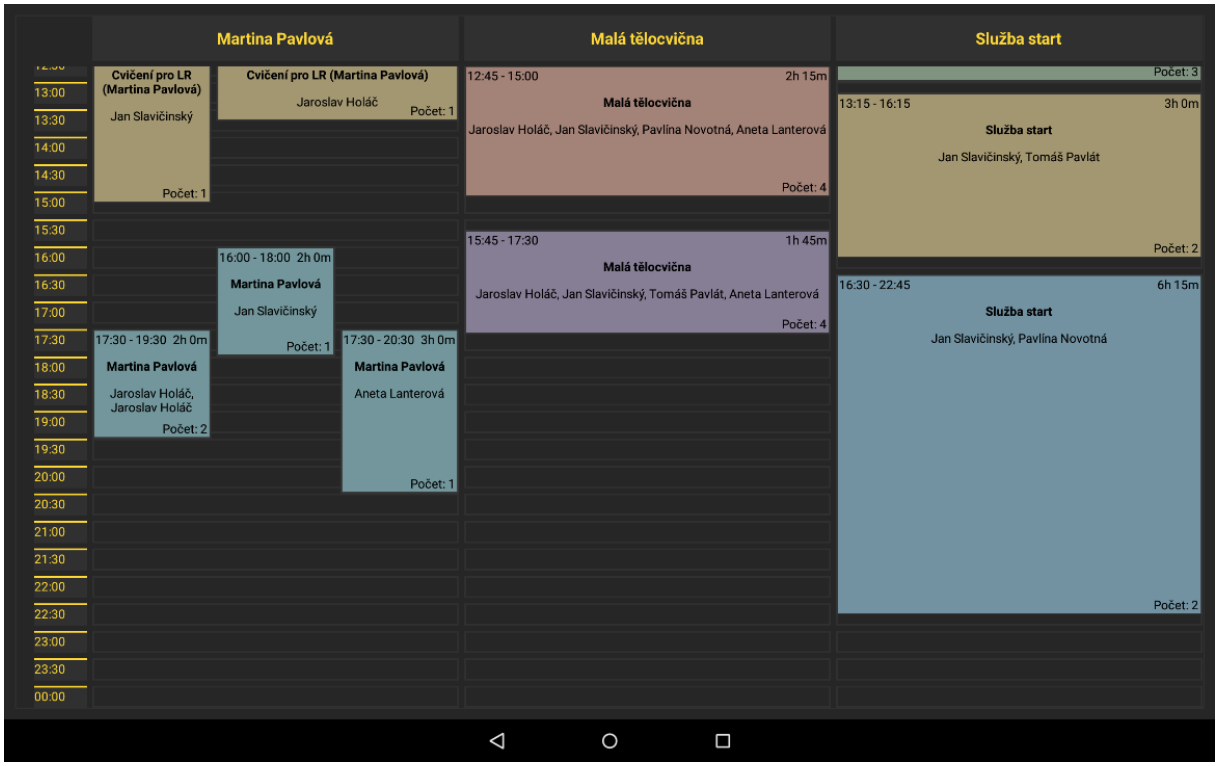
10 Výsledná aplikace a její použití

Tato kapitola slouží pro představení implementované multiplatformní aplikace, převážně z pohledu uživatelského rozhraní, jelikož její funkcionalita již byla podrobně popsána v předchozích kapitolách, není proto potřeba jednotlivé funkcionality aplikace znovu popisovat a detailněji rozebírat.

Všechny obrázky uživatelského rozhraní v této kapitole zobrazují pouze vzhled aplikace na platformě Android. V aplikaci pro iOS a Windows 10 prvky vypadají trochu jinak, ovšem jedná se pouze o odlišnou vizualizaci, funkčně se chovají naprosto stejně jako u platformy Android.

10.1 Vzhled výsledné aplikace

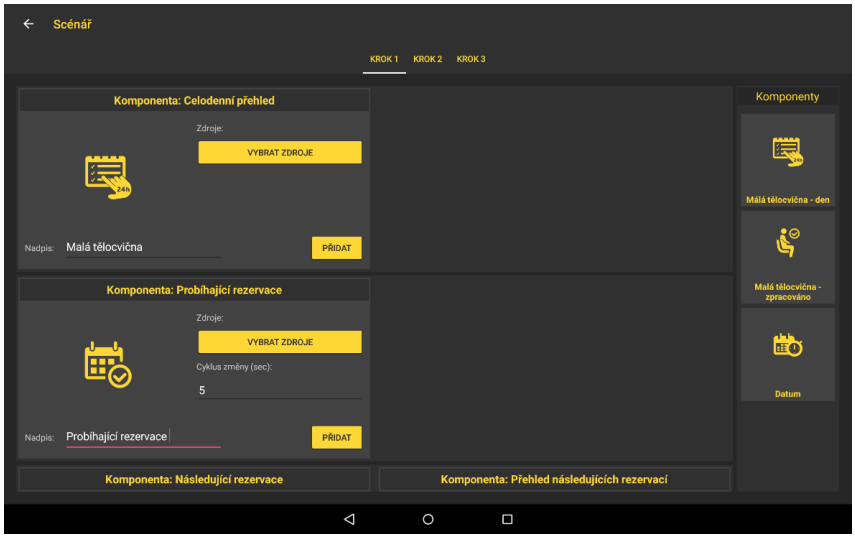
Na hlavní obrazovce aplikace se promítají jednotlivé scénáře s okny, které jsou přepínány v zadaném intervalu při jejich vytváření. Součástí této obrazovky je i menu aplikace, které je přístupné po stisknutí tlačítka myši nebo dotykové obrazovky na jakékoliv místo na hlavní obrazovce. Pomocí menu se lze v aplikaci navigovat např. do aplikačního nastavení nebo na obrazovku pro vytváření scénářů.



Obrázek 26: Hlavní obrazovka aplikace pro zobrazování scénářů.

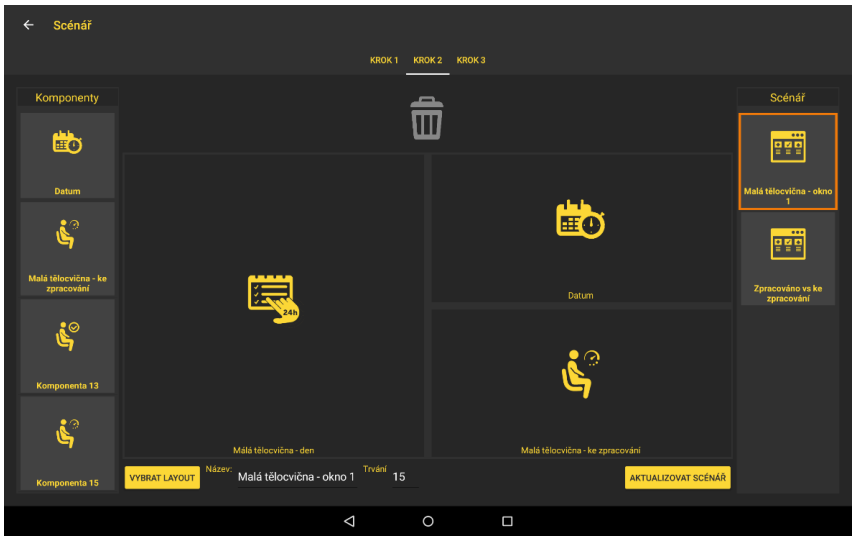
10.1.1 Vytváření scénářů

Pomocí navigačního menu na hlavní stránce, lze otevřít obrazovku pro vytváření scénářů na základě třech kroků. Na následujícím obrázku můžeme vidět první krok scénáře, který znázorňuje vytváření a přidávání komponent do seznamu pro pozdější použití.



Obrázek 27: První krok pro vytváření scénářů.

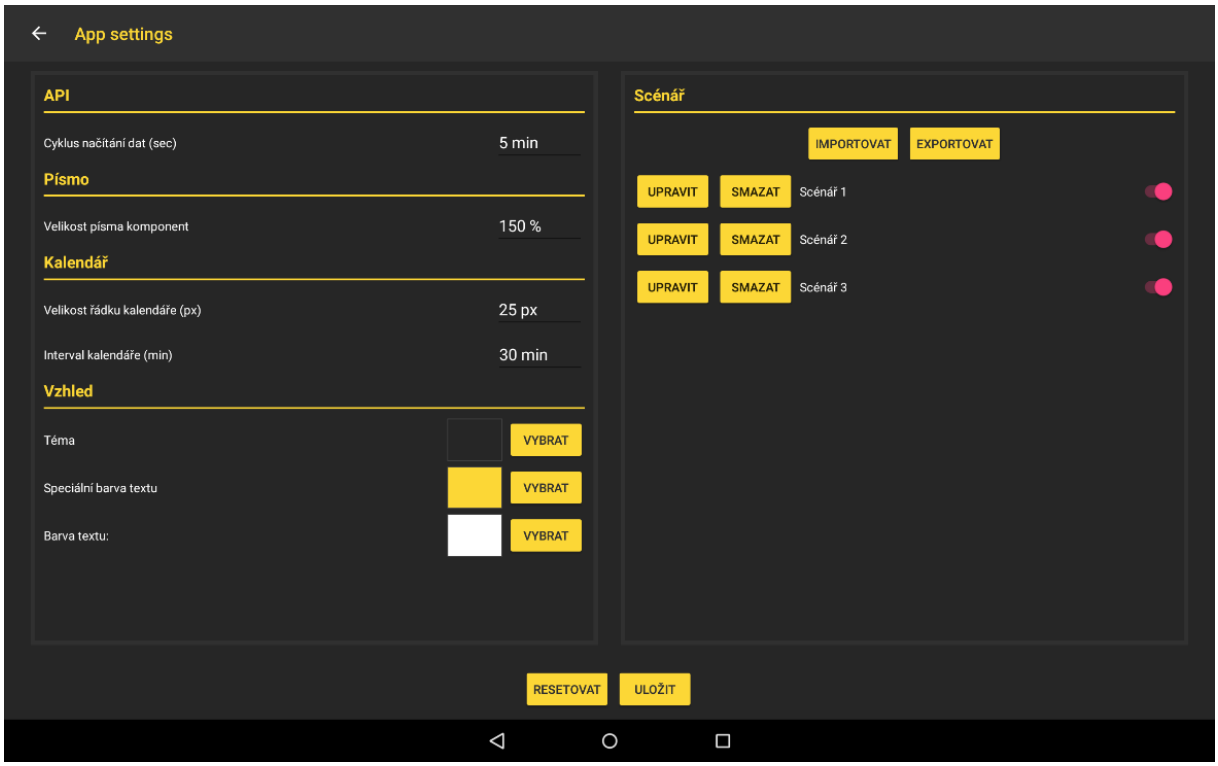
Druhým krokem je již samotné vytváření jednotlivých oken, které jsou součástí scénáře. Pomocí seznamu vybraných komponent (vytvořených v prvním kroku) je možné vkládat komponenty do polí určených na základě výběru vhodného vzhledu aktuálního okno. Následně po přidání samotného okna scénáře, je toto okno zobrazeno v pravé části obrazovky, v seznamu oken.



Obrázek 28: Druhý krok pro vytváření scénářů.

10.1.2 Nastavení aplikace

Po vyplnění validních údajů k přihlášení je uživatel přesměrován na obrazovku aplikačního nastavení, na které lze měnit nastavení pro celou aplikaci nebo pouze pro jednotlivé komponenty. Na levé straně obrazovky zde můžeme najít např. nastavení pro velikost písma, cyklus pro aktualizaci dat nebo natavení vzhledu celé aplikace jako je barva písma a pozadí. Na pravé straně jsou následně možnosti týkající se práce se scénáři (import, export, aktualizace, atd.).



Obrázek 29: Obrazovka aplikačního nastavení.

10.2 Použití aplikace

Aplikace je určena především provozovatelům rezervačních systémů a slouží k prezentaci a zobrazení aktuálních informací o rezervacích. Pro ukázku je v současné době aplikace napojena na rezervační systém běžící na platformě reenio `https://reenio.cz/`.

Aplikace umožňující prezentaci informací o aktuálním stavu rezervací. Na základě konfigurace je možné specifikovat prezentační scénáře, které kombinují informace o rezervacích a případný marketingový obsah.

Díky aplikaci je možné zobrazovat např. který zákazník hraje na konkrétním kurtu, jaké je pořadí pro rezervované zákazníky apod. Doporučený způsob provozu této aplikace je v kombinaci s televizní obrazovkou či panelem umístěným přímo v čekárně nebo v místě poskytování služeb.

11 Zhodnocení platformy

Před samotnou implementací jsem se rozhodoval jestli použít framework Xamarin.Forms nebo využít nativní vývoj pomocí Xamarinu. Na základě očekávaných vlastností aplikace jsem zvolil Xamarin.Forms, protože multiplatformní aplikace měla klást důraz především na aplikační logiku a zpracování dat. Naopak specifické funkcionality, jako je využití Bluetooth, přehrávání hudby nebo zobrazování notifikací, které by bylo potřeba implementovat zvlášť pro každou platformu, nebyly požadavkem na aplikaci.

Na základě požadavků na aplikaci byl vybrán právě framework Xamarin.Forms, který je pro implementovanou aplikaci nejlepší volbou. Díky tomuto frameworku se čas na implementaci velice zkrátil, protože nebylo potřeba implementovat uživatelské rozhraní a jiné třídy zvlášť pro každou platformu. Jedinými rozdílnými prvky při implementaci mezi platformami jsou chybějící funkcionality, které se dají implementovat pomocí tzv. rendererů a service, zvlášť pro každou cílovou platformu. Součástí frameworku Xamarin.Forms jsou různé výhody i nevýhody. Při vývoji aplikace jsem na některé z nich narazil.

Výhody

Hlavní výhodou tohoto frameworku je multiplatformnost neboli sdílení jedné funkcionality pro všechny podporované platformy (Android, iOS, UWP). Jedná se především o sdílení uživatelského rozhraní, datového modelu a aplikační logiky. Z těchto důvodů nemusí vývojář umět vyvíjet zvlášť pro každou platformu, ale stačí mu pouze framework Xamarin.Forms a může vyvíjet nativní aplikace pro všechny tři operační systémy.

Implementovaná aplikace BoardReservations obsahuje 99% sdíleného kódu. Pouze pro vypnutí aplikace, práci se soubory a proces Drag and Drop jsou implementovány zvlášť pro každou platformu, jelikož je samotné framework Xamarin.Forms nepodporuje. Velkou výhodu, kterou jsem využil při implementaci multiplatformní aplikace, je možnost vývoje v prostředí Visual Studio, který je jeden z nejlepších vývojových nástrojů.

Nevýhody

Pro začínající vývojáře na tomto frameworku může být velkou nevýhodou samotná tvorba uživatelských rozhraní, protože neobsahuje žádný návrhový GUI nástroj. Tato nevýhoda přináší uživateli velkou starost, protože musí aplikaci zkompilovat do zařízení nebo do emulátoru, aby zjistil vzhled aplikace pro danou platformu. Ve Visual Studio 2017 sice existuje možnost pro náhled přímo při implementaci (Xamarin.Forms Previewer), ale bohužel tato funkce ještě není úplně doladěná a nefunguje úplně správně, proto je stále lepší zkompilovat aplikaci přímo do zařízení.

Jednou z nevýhod na které jsem při implementaci narazil, je možnost využití omezeného počtu UI prvků, kdy zcela chybí např. checkbox. Tuto problematiku jsem vyřešil pomocí vytvo-

ření vlastních prvků, které jsou sice časově náročnější na implementaci, ale následně podporují vlastní funkcionalitu.

Při vývoji jsem se také setkával s "bugy", které jsou přímo v jádru frameworku. Například, když jsem se snažil vnořit uživatelské tlačítko do několika vrstev, které byly tvořené prvkem StackLayout, tlačítko na platformě Android nereagovalo na událost klik. Takového chyby lze hlásit přímo na fórum Xamarinu a v následujících verzích frameworku (vychází skoro každý týden) bývá velká část z nich opravena.

12 Závěr

Cílem této práce bylo přiblížit různé možnosti multiplatformního vývoje pro mobilní aplikace, ale především detailněji prozkoumat a popsat multiplatformní vývoj z prostředí Xamarin. Následně pak na základě zjištěných informací implementovat aplikaci, kterou bude možné spustit na systému Android nebo iOS.

Výsledkem této práce je implementovaná aplikace BoardReservations, která slouží jako přehled rezervačního systému na obrazovkách s operačním systémem. Multiplatformní aplikace je složena na front-endu z několika dynamických obrazovek na základě uživatelských nastavení a na back-endu z logické implementace pro tyto obrazovky a jejich komunikace s API webovým rozhraním.

Na aplikaci byl znázorněn proces tvorby mobilních aplikací, od návrhu struktury (wireframe), přes výběr vhodného multiplatformního vývojového prostředí, až po samotnou implementaci multiplatformní aplikace.

Pro představení různých multiplatformních technologií byly stručně popsány technologie Apache Cordova a Titanium. Pro samotnou aplikaci bylo požadavkem vývoj v prostředí Xamarin. Následně jsem toto multiplatformní vývojové prostředí velmi detailně prozkoumal a nakonec jsem pro samotnou implementaci aplikace vybral jeho framework, který se nazývá Xamarin.Forms. Vlastnosti tohoto frameworku, které mě přesvědčily pro výběr této technologie jsou především možnost sdílení převážné většiny kódu mezi platformami a zároveň oddělení uživatelské vrstvy, business vrstvy a datového modelu pomocí návrhového vzoru MVVM.

Při implementaci multiplatformní aplikace v prostředí Xamarin.Forms se projevila i jedna z velkých nevýhod tohoto frameworku. Jedná se konkrétně o její chybovost uvnitř samotného jádra frameworku. Jelikož se platforma Xamarin stále vyvíjí, vzniklé chyby uvnitř systému se s každou novou vydanou verzí opravují, ale můžou zde vznikat další a nové. Při implementaci jsem také narazil na nevýhodu samotných operačních systémů, kterou jsem objevil až při implementaci komponenty videa. Jedná se o zákaz, v jádru systémů, spouštět automaticky videa pomocí webového prohlížeče. Z tohoto důvodu není v aplikaci implementována komponenta pro přehrání videa na základě url odkazu.

Součástí této práce bylo i nasazení aplikace do reálného světa. Z tohoto důvodu jsou zde i popsány procesy pro publikace aplikace pro platformu Android a iOS. Následně je pak zhodnocena implementovaná multiplatformní aplikace a celá platforma Xamarin.Forms.

Literatura

[1] *Global mobile OS market. Statista [online]. 2017 [cit. 2018-02-13]. Dostupné z:* https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[2] *What Is Google Android?. Lifewire [online]. 2017 [cit. 2018-02-15]. Dostupné z:* https://www.lifewire.com/what-is-google-android-1616887

[3] *Android - 8.0 Oreo. Android [online]. 2017 [cit. 2018-02-15]. Dostupné z:* https://www.android.com/versions/oreo-8-0/

[4] *I want to develop Android Apps. Android Authority [online]. 2017 [cit. 2018-02-15]. Dostupné z:* https://www.androidauthority.com/develop-android-apps-languages-learn-391008/

[5] *Android - mobile platform. Android Developers [online]. 2017 [cit. 2018-02-15]. Dostupné z:* https://developer.android.com/about/android.html

[6] *In: Mobile App Development - iOS and Android [online]. 2017 [cit. 2018-02-15]. Dostupné z:* https://cayland.com/

[7] *Apple iOS. TechTarget [online]. 2017 [cit. 2018-02-18]. Dostupné z:* http://searchmobilecomputing.techtarget.com/definition/iOS

[8] *Write iOS Apps. lifehacker.com [online]. 2017 [cit. 2018-02-18]. Dostupné z:* https://lifehacker.com/i-want-to-write-ios-apps-where-do-i-start-1644802175

[9] *Publikování aplikací Android. Microsoft Docs [online]. 2017 [cit. 2018-02-18]. Dostupné z:* https://docs.microsoft.com/cs-cz/xamarin/ios/deploy-test/app-distribution/app-store-distribution/publishing-to-the-app-store?tabs=vsmac

[10] *Windows 10 Mobile. TheVerge [online]. 2015 [cit. 2018-02-22]. Dostupné z:* https://www.theverge.com/2015/4/30/8525057/windows-10-mobile-release-date

[11] *Servisní vývoj Windows 10 Mobile pokračovat bude. Cnews.cz [online]. 2015 [cit. 2018-02-22]. Dostupné z:* https://www.cnews.cz/proc-pokracuje-vyvoj-testovani-windows-10-mobile/

[12] *Windows 10 SDK. Microsoft Docs [online]. 2017 [cit. 2018-02-22]. Dostupné z:* https://developer.microsoft.com/cs-cz/windows/downloads/windows-10-sdk

[13] *Choosing a programming language. Microsoft Docs [online]. 2017 [cit. 2018-02-22]. Dostupné z:* https://docs.microsoft.com/en-us/windows/uwp/porting/getting-started-choosing-a-programming-language

[14] *Distribute apps - Microsoft Store. Microsoft Docs [online]. 2017 [cit. 2018-02-22]. Dostupné z:* https://docs.microsoft.com/en-us/microsoft-store/distribute-apps-to-your-employees-microsoft-store-for-business

[15] *In: 5 Reasons to Upgrade to Windows 10 Now [online]. 2017 [cit. 2018-02-22]. Dostupné z:* http://blog.computechlimited.com/2017/07/19/5-reasons-to-upgrade-to-windows-10-now/

[16] *Úvod do pro vývoj mobilních řešení - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-02-25]. Dostupné z:* https://docs.microsoft.com/cs-cz/xamarin/cross-platform/get-started/introduction-to-mobile-development

[17] *Principy Xamarin mobilní platformy. Microsoft Docs [online]. 2017 [cit. 2018-02-25]. Dostupné z:* https://docs.microsoft.com/cs-cz/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform

[18] *In: Cross Platform Applications using Visual Studio [online]. 2017 [cit. 2018-02-25]. Dostupné z:* https://www.mill5.com/cross-platform-applications-using-visual-studio/

[19] *Overview - Apache Cordova. Apache Cordova [online]. 2015 [cit. 2018-02-27]. Dostupné z:* http://cordova.apache.org/docs/en/4.0.0/guide/overview/index.html#Overview

[20] *Learn about mobile development with JavaScript in Visual Studio 2017. Microsoft Docs [online]. 2016 [cit. 2018-02-27]. Dostupné z:* https://docs.microsoft.com/en-us/visualstudio/cross-platform/tools-for-cordova/?view=toolsforcordova-2017

[21] *In: Installing and setting up Apache Cordova [online]. 2017 [cit. 2018-02-27]. Dostupné z:* https://www.samirkamble.com/installing-and-setting-up-apache-cordova-phonegap-environment/

[22] *Appcelerator Open Source. Appcelerator [online]. 2018 [cit. 2018-03-02]. Dostupné z:* https://www.appcelerator.org/#titanium

[23] *Titanium Platform Overview. Appcelerator [online]. 2018 [cit. 2018-03-02]. Dostupné z:* http://docs.appcelerator.com/platform/latest/#!/guide/Titanium_Platform_Overview

[24] *Axway Appcelerator Studio - Documentation and Guides. Appcelerator [online]. 2018 [cit. 2018-03-02]. Dostupné z:* http://docs.appcelerator.com/platform/latest/#!/guide/Axway_Appcelerator_Studio

[25] *Architektura - Xamarin. Microsoft Docs [online]. 2018 [cit. 2018-03-03]. Dostupné z:* https://developer.xamarin.com/guides/android/under_the_hood/architecture/

[26] *IOS architektura - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-03-06]. Dostupné z:* https://developer.xamarin.com/guides/ios/under_the_hood/architecture/

[27] *Introduction to Xamarin.iOS for Visual Studio - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-03-06]. Dostupné z:* https://docs.microsoft.com/en-us/xamarin/ios/get-started/installation/windows/introduction-to-xamarin-ios-for-visual-studio

[28] *In: Xamarin archives [online]. 2017 [cit. 2018-03-06]. Dostupné z:* https://mobilefirstcloudfirst.net/category/xamarin/

[29] *Sdílení projektů - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-03-07]. Dostupné z:* https://developer.xamarin.com/guides/cross-platform/application_fundamentals/shared_projects/

[30] *HERMES, Dan. Xamarin Mobile Application Development. 1. California: Apress, 2015, s. 382-385. ISBN 9781484202159.*

[31] *Úvod do knihovny přenosných tříd - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-03-08]. Dostupné z:* https://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/

[32] *Standardní rozhraní .NET. Microsoft Docs [online]. 2017 [cit. 2018-03-08]. Dostupné z:* https://developer.xamarin.com/guides/cross-platform/application_fundamentals/net-standard/

[33] *In: Možnosti sdílení kódu - Xamarin [online]. 2017 [cit. 2018-03-08]. Dostupné z:* https://developer.xamarin.com/guides/cross-platform/application_fundamentals/code-sharing/

[34] *Functional and non-functional requirements. Quora [online]. 2017 [cit. 2018-03-09]. Dostupné z:* https://www.quora.com/What-are-the-functional-and-non-functional-requirements-in-software-engineering

[35] *In: How to use API [online]. 2017 [cit. 2018-03-015]. Dostupné z:* https://snipcart.com/blog/apis-integration-usage-benefits

[36] *Entity Data Model klíčové koncepty. Microsoft Docs [online]. 2017 [cit. 2018-03-10]. Dostupné z:* https://msdn.microsoft.com/cs-cz/library/ee382840(v=vs.110).aspx

[37] *MVVM - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-03-17]. Dostupné z:* https://developer.xamarin.com/guides/xamarin-forms/enterprise-application-patterns/mvvm/

[38] *PETZOLD, Charles. Creating Mobile Apps with Xamarin.Forms. 1. Redmond, Washington: Microsoft Press, 2015, s. 491-534. ISBN 978-1-5093-0297-0.*

[39] *MessagingCenter - Xamarin. Microsoft Docs [online]. 2017 [cit. 2018-03-17]. Dostupné z:* https://developer.xamarin.com/guides/xamarin-forms/application-fundamentals/messaging-center/

[40] *Strategy Design Pattern - C#. DotNetTricks [online]. 2017 [cit. 2018-03-17]. Dostupné z:* http://www.dotnettricks.com/learn/designpatterns/strategy-design-pattern-c-sharp

[41] *What is XAML?. MSDN [online]. 2018 [cit. 2018-03-19]. Dostupné z:* https://msdn.microsoft.com/en-us/library/cc295302.aspx

[42] *PETZOLD, Charles. Creating Mobile Apps with Xamarin.Forms. 1. Redmond, Washington: Microsoft Press, 2015, s. 418-457. ISBN 978-1-5093-0297-0.*

[43] *The Command Interface. Microsoft Docs [online]. 2018 [cit. 2018-03-19]. Dostupné z:* https://docs.microsoft.com/en-us/xamarin/xamarin-forms/app-fundamentals/data-binding/commanding

[44] *Introduction to the C# Language and the .NET Framework. Microsoft Docs [online]. 2015 [cit. 2018-03-20]. Dostupné z:* https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

[45] *JSON : jednotný formát pro výměnu dat. Zdrojak.cz [online]. 2008 [cit. 2018-03-20]. Dostupné z:* https://www.zdrojak.cz/clanky/json-jednotny-format-pro-vymenu-dat/

A Použité technologie

A.1 XAML

Programovací jazyk XAML je zkratka pro Extensible Application Markup Language. Jedná se o modifikaci jazyka XML pro .NET framework, který je využívaný k vytvoření grafického rozhraní v aplikacích nové generace. Ve frameworku Xamarin.Forms se tento jazyk využívá pro implementaci View vrstvy z návrhového vzoru MVVM. XAML, jako celek je tvořen soubory s příponou .xaml a .xaml.cs. Na rozhraní .xaml souboru se vytváří uživatelské rozhraní psané právě pomocí syntaxí jazyka XAML. A v souboru s příponou .cs je tvořen tzv. code-behind, který je následně propojen s uživatelských rozhraním pomocí tzv. Data bindingu. Současně vše, co lze vytvořit pomocí jazyku XAML, je také možné vytvořit i pomocí .NET jazyků C# nebo VB.NET.[41]

Výhody

- Pro návrh uživatelského rozhraní je daleko více čitelnější a intuitivnější než kód, je to dáno především strukturou předek-potomek
- Grafické prvky lze psát ručně pomocí kódu nebo pomocí grafických nástrojů

Nevýhody

- Nemůže obsahovat žádný kód, pouze v code-behind
- Nemůže volat žádné metody, pouze vyvolávat nějakou událost (např. command)
- Nelze vytvořit instanci třídy s parametrickým konstruktorem

A.1.1 Data binding

DataBinding je prostředek, který byl vytvořen právě pro jazyk XAML a MVVM vzor. Tento prostředek umožňuje vytvořit vazbu pro výměnu dat mezi View a ViewModelem. S daty lze pracovat buď přímo na ViewModelu, kde jsou data následně překreslena, nebo aktualizována na View nebo stejným způsobem, ale v opačném směru, kdy jsou aktualizovaná data na View přenášena do ViewModelu. O tento proces přenosu dat se stará právě technologie Data binding, která reálně představuje vztah mezi cílem a zdrojem neboli BindableProperty (View) a konkrétní Property (ViewModel), kde může existovat již zmíněná jednosměrná nebo obousměrná vazba. Vazbu pro binding lze definovat těmito 4 způsoby:

- Default – jedná se o základní hodnotu, která je nastavena pro Bindable Property jako výchozí
- OneWay – data jsou přenášeny směrem od zdroje k cíli

- OneWayToSource – data jsou přenášeny z cíle ke zdroje
- TwoWay – data jsou směrovaná oběma směry[42]

```
<Entry Text="{Binding LastLoadedFormatted,Mode=TwoWay}"
       VerticalOptions="Center" HorizontalOptions="Center" />
```

Výpis 19: Ukázka Entry Bindingu pro Property Text.

A.1.2 Command

Příkazy se definují především v rámci architektury MVVM a jsou vyvolávány ze XAMLu. Namísto klasických event handlerů definovaných v code-behind, které jsou složité na testování, se v MVVM využívají právě tyto příkazy. Jedná se o označení třídy, která implementuje rozhraní ICommand.

```
public interface ICommand
{
   event EventHandler CanExecuteChanged;
   bool CanExecute(object parameter);
   void Execute(object parameter);
}
```

Výpis 20: Rozhraní pro Command

Rozhraní ICommand obsahuje následující dvě metody a jednu událost:

- Execute metoda obsahuje implementaci, která se provádí, když se zavolá právě tento command.
- CanExecute metoda vrací informaci, zda může být daný command spuštěn.
- CanExecuteChanged je událost, která se vyvolává, jestliže v metodě CanExecute dojde ke změně.[43]

A.2 C#

Vysokoúrovňový objektově orientovaný programovací jazyk C# byl vyvinut společností Microsoft zároveň s frameworkem .NET. Jeho nepřímým potomkem je jazyk C, jelikož jazyk vznikl na základech jazyků C++ a Javy, pravě programovací jazyk C++ je následovník jazyka C. Jazyk C# lze hlavně využívat k tvorbě desktopových aplikací, webových služeb, databázových programů nebo pro vývoj aplikací. Právě spojením tohoto jazyka a jazyka XAML, vznikl nový programovací styl, který se využívá při vývoji v Xamarin.Forms nebo ve WPF aplikací.[44] Mezi základní vlastnosti jazyku patří zejména:

- C# nepodporuje vícenásobnou dědičnost a to způsobuje, že každá třída může mít pouze jednoho předka. Pomocí rozhraní lze tuto potřebu částečně obejít, jelikož třída může implementovat libovolný počet rozhraní.
- Neobsahuje žádné globální proměnné a metody, vše musí být deklarováno uvnitř třídy. Nahradit tento princip lze pomocí použití statických tříd a deklarací statickým proměnných a metod.
- Jazyk je case sensitive, to znamená, že rozlišuje mezi velkými a malými písmeny.

A.3 JSON

JSON je zkratka pro JavaScript Object Notation, jedná se o způsob zápisu a přenosu dat, který je nezávislý na počítačové platformě. JSON je převážně určen pro přenos dat, která můžou být organizovaná v polích nebo i v objektech. Vstupním prvek může být jakákoliv datová struktura (řetězec, číslo, boolean, objekt nebo i vlastní datový typ) a výstupním prvkem je následně vždy řetězec. JSON je zcela nezávislým formátem dat a je velmi snadno čitelný a zapisovatelný.[45] Tato technologie bude použita při přenosu dat mezi API rozhraním a aplikací. A dále také jako uchovávání jednotlivých scénářů, které bude možné ukládat na interní úložiště.

```
{"person":
   {"name": "Robin",
   "age": 35},
   "met": false,
   "hobbies": null,
   "data": [1,2]}
}
```

Výpis 21: Ukázka třídy Person ve formátu JSON

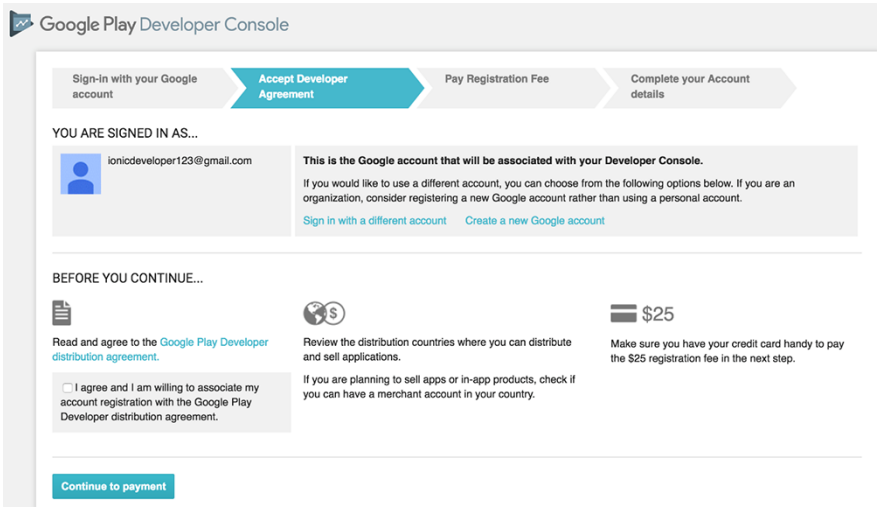
A Publikace aplikace

A.1 Proces nasazení aplikace pro Android

A.1.1 Založení Google Play účtu

Pro publikování aplikací pro platformu Android, jak již bylo dříve zmíněno, je zapotřebí zaregistrovat si a zaplatit účet pro vývojáře, skrze který bude možné vystavit aplikaci pro veřejnost. V následujících krocích bude popsán proces vytvoření takového účtu.

1. Registrace nového účtu se provádí na url `https://play.google.com/apps/publish/signup/`
2. Pomocí tlačítka "vytvořit nový účet Google"se otevře nové okno a po vyplnění osobních údajů (jméno, příjmení, uživatelské jméno, heslo, atd.) se vytvoří standardní Google účet.
3. Přihlášení k vytvořenému účtu a odsouhlasení Google podmínek následuje přechod k platební bráně.
4. Zde je potřeba zadat číslo platební karty a údaje potřebné k potvrzení částky 25 USD.
5. Po potvrzení platby je možné v systému vyplnit informace o identitě vývojáře nebo společnosti (název, adresa, email, apod.).
6. Vyčkání na aktivaci účtu, jelikož spárování nového účtu s Google Play systémem může trvat až 48 hodin.



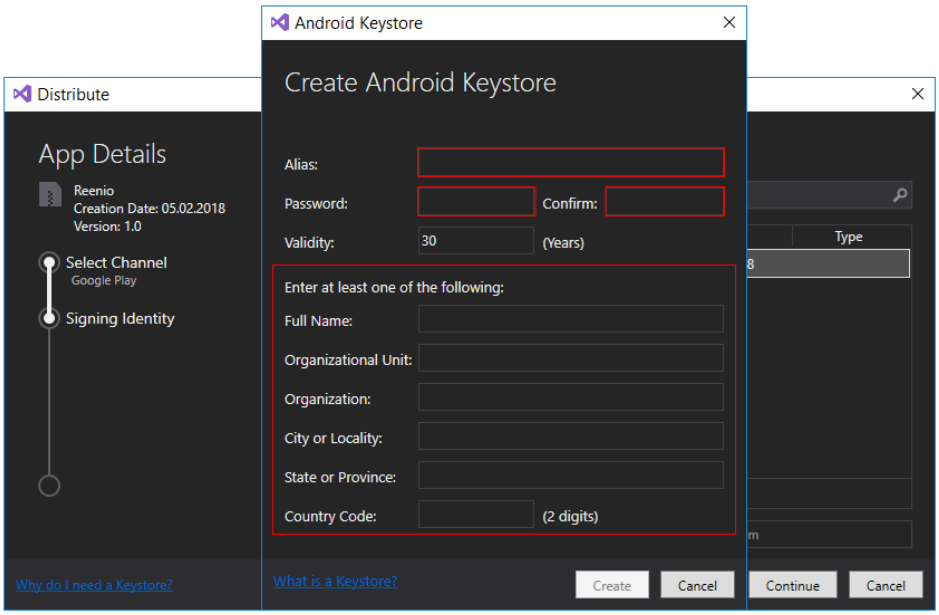
Obrázek 30: Založení Google Play účtu pomocí webu.

A.1.2 Podepisující certifikát

Jakákoliv aplikace umístěná na Google Play musí být podepsána vlastním podepisujícím certifikátem (tzv. Android KeyStore) a následně i každá jeho nová verze. Tato podmínka vznikla z

důvodu lepšího zabezpečení aplikací a to především z vývojářského pohledu, aby nedocházelo k vykrádání aplikací a jejich neoprávněnou publikací. Jednou z možností jak vytvořit Android KeyStore je pomocí externího nástroje např. v Android Studiu nebo přímo ve Visual Studiu. Proces vytvoření klíče v prostředí Visual Studio je popsán v následujících krocích.

1. Z kontextového menu na projektu je potřeba zvolit archivaci.
2. Počkat na zkompilování projektu a stisknout tlačítko Distribuce.
3. Pomocí tlačítka plus přidat podepisující identitu.
4. Vyplnit povinné údaje formuláře pro nový podepisující certifikát (název, heslo, dobu platnosti, osobní údaje firmy nebo fyzické osoby, atd.).



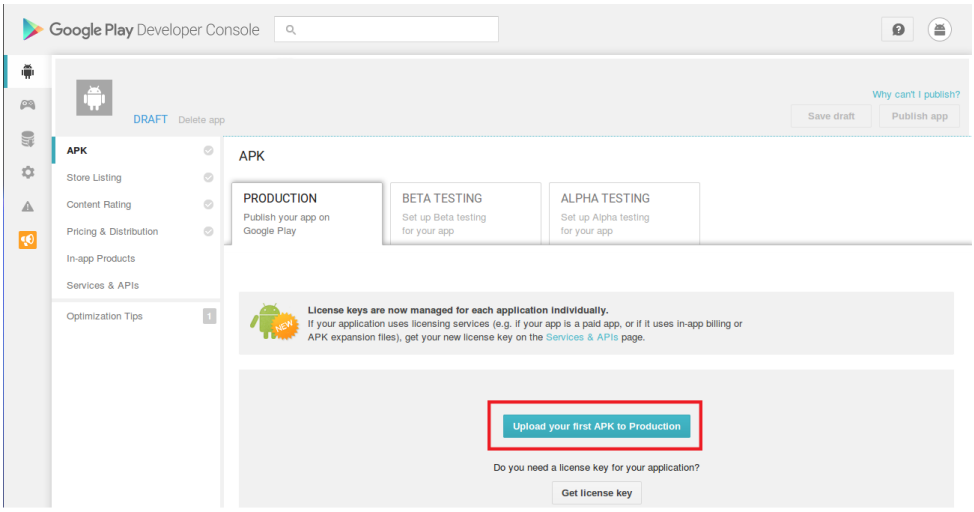
Obrázek 31: Vytvoření certifikátu pomocí Visual Studio.

A.1.3 Vystavení aplikace na Google Play

Po předchozích procesech nutných k vystavení aplikace do systému Google Play, je nyní již možné aplikaci zveřejnit pro uživatele Android systému. Pomocí následujících kroků si ukážeme jakým způsobem je potřeba aplikaci zkompilovat, aby vše proběhlo v pořádku a nemuseli jsme řešit případné potíže.

1. V prostředí Visual Studio je potřeba v Android projektu nastavit kompilaci na Release.
2. Vybrat archivaci z kontextového menu na projektu.
3. Pomocí tlačítka Distribuce vyvolat průvodce nasazení aplikace.

4. Zvolit kanál Google Play.
5. Přiřadit správný podepisující certifikát k aplikaci.
6. Nastavit Google API a zadat správné heslo k podepisujícímu certifikátu.
7. Nahrání aplikace do produkce skrz větev Production a stisknutí tlačítka Upload.
8. Jestliže nahrání aplikace proběhlo v pořádku, je posledním krokem publikace přímo na web. Toto lze nastavit po přihlášení k webu `https://play.google.com/store` ve správě aplikace (zahájit vydávání v produkčním kanálu). V případě, že neproběhlo nahrání aplikace v pořádku, zobrazí se u distribuce červená ikonka křížku s informacemi ohledně dané chyby.
9. Vyčkání na automatické otestování aplikace.
10. Po úspěšném zpracování a nasazení bude u nasazené aplikace popisek "Publikováno".



Obrázek 32: Publikování Android aplikace na webu Google Play.

A.2 Proces nasazení aplikace pro iOS

A.2.1 Založení Apple Store účtu

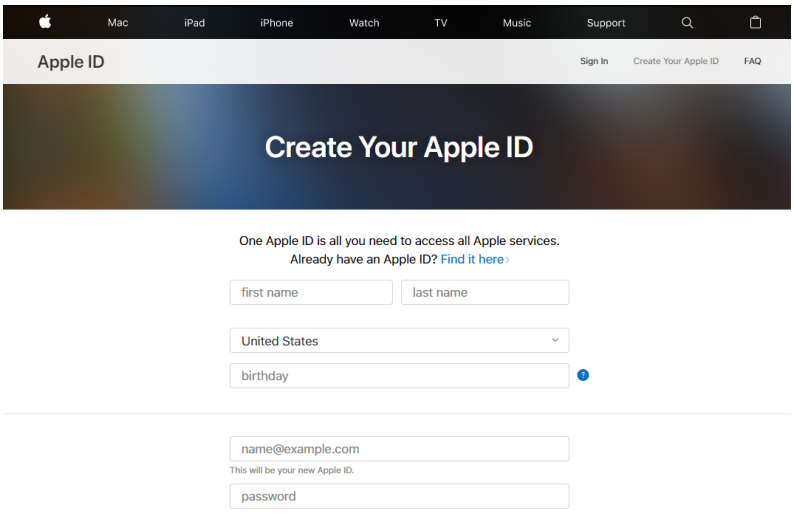
Vytvoření účtu a propojení se službou Apple Developer Program dává vývojáři možnost publikovat své aplikace v prostředí App Store. Následující kroky popisují jak takový účet vytvořit.

1. Nejprve je potřeba přihlásit se pomocí standardního Apple účtu na následující webové adrese `https://developer.apple.com/`

2. Zde po přihlášení uživatele je potřeba zahájit proces přihlášení k Apple Developer Program. Stačí pouze na hlavní obrazovce stisknout tlačítko s odkazem Join the Apple Developer Program.

3. Následně je potřeba vyplnit osobní údaje jednotlivce nebo organizace.

4. Posledním krokem je přechod na platební bránu a zaplacení ročního poplatku ve výši 99 USD pomocí platební karty. Pro udržení developer členství je potřeba tento poplatek platit každý rok.



Obrázek 33: Založení Apple ID.

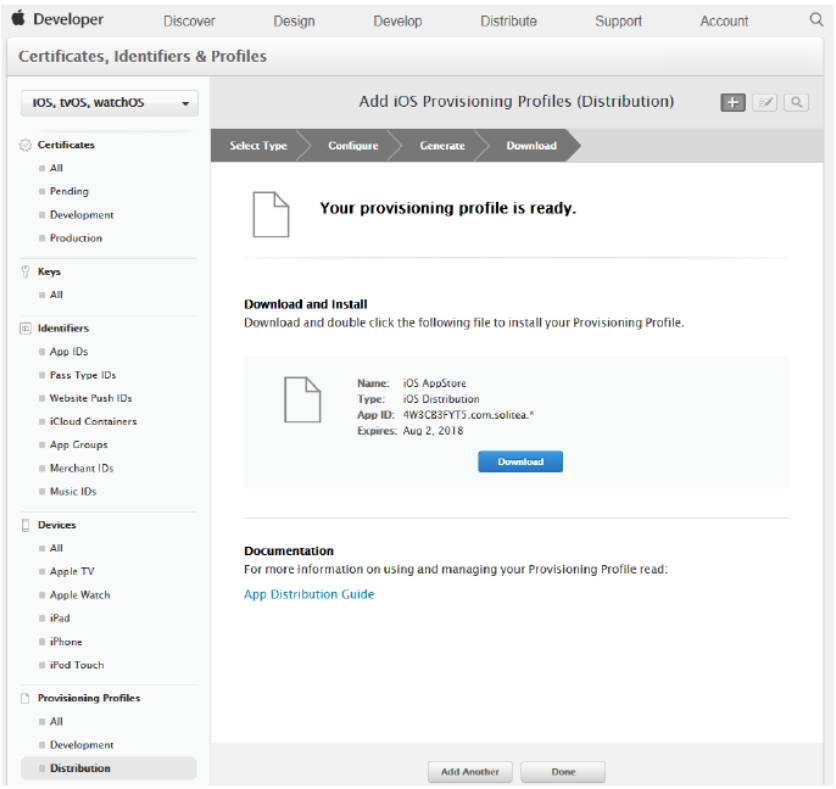
A.2.2 Certifikát a distribuční profil

Každou aplikaci pro platformu iOS je nutné zabezpečit a podepsat Apple certifikátem, který se nejprve musí vygenerovat pomocí distribučního profilu. Následující postup popisuje daný proces.

1. V sekci Certificaties, Ids & Profiles, která je přístupná po přihlášení k vývojářskému účtu je potřeba přidat nový certifikát.

2. Zadáním názvu a unikátního čísla (Bundle ID) se následně vygeneruje certifikát.

3. Následně je potřeba vytvořit distribuční profil v sekci Provisioning > Distribution.
4. Pomocí tlačítka zvolit proces nového profilu.
5. Zde vložit App ID a certifikát, který byl vygenerován v předchozích krocích.
6. Zadat název nového profilu a stisknout tlačítko pro generování profilu.
7. Následně se vygeneruje profil pro sestavení IPA balíčku, který se přiloží k projektu v jeho vlastnostech.



Obrázek 34: Vytvoření distribučního profilu.

A.2.3 Konfigurace pro iTunes Connect

Před samotným nahráním aplikace do App Store je potřeba, aby aplikace byla správně nastavena a registrovaná ve službě iTunes Connect. Tato služba představuje sadu webových nástrojů společnosti Apple, mimo jiné i pro správu mobilních aplikací v App Store. Následující kroky budou znázorňovat proces registrace aplikace.

1. Nejprve je potřeba se přihlásit na webové stránce `https://itunesconnect.apple.com/login`
2. Po přihlášení otevřít na hlavní stránce sekci My Apps.

3. Přidat pomocí New App novou aplikaci.

4. Vyplnit formulář specifikující název aplikace, platformu iOS, Bundle ID a výchozí jazyk.

5. Dále je potřeba vyplnit detaily aplikace jako jsou popis, kategorie a klíčová slova, podle kterých lze aplikaci najít.

6. V sekci Pricing and Availability specifikovat cenu aplikace.

7. V sekci Prepare for Submission nadefinovat potřebné komponenty pro úspěšnou publikaci. Jedná se především o ikonu aplikace a snímky obrazovky pro iPhone, iPad apod..

8. Posledním krokem je potřeba vyplnit Rating a App Review Information. Zde se zadávají informace pro schvalovací proces aplikace. Je potřeba vyplnit stručný popis aplikace a její funkční proces. Jestliže je pro spuštění aplikace potřeba přihlašovací údaje, je nutné tyto údaje zpřístupnit ke schvalovacímu procesu. V případě nejasností ze strany Applu při schvalovacím procesu, je vhodné mít vyplněný kontakt, pro jejich případné vyjasnění.

A.2.4 Kompilace IPA souboru

Pro samotnou publikaci je potřeba zkompilovat projekt do balíčku s příponou IPA. Tento balíček je následně potřeba podepsat profilem pro App Store, který byl vytvořen výše. Kompilace a podpis se provádí následujícími kroky.

1. Otevřít vlastnosti projektu pomocí kontextového menu .

2. Zde je potřeba nastavit konfiguraci projektu na Ad-Hoc, případě že Ad-Hoc není přístupný vybrat Release.

3. Nastavit název balíčku a uložit nastavení.

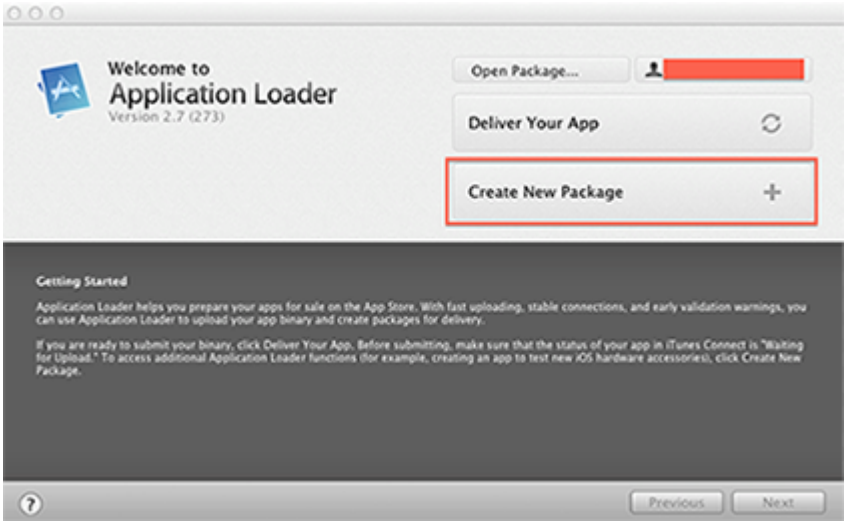
4. Na projektu je následně potřeba spustit kompilaci aplikace. V případě provádění kompilace na Windows zařízení, se zobrazí okno, kde je potřeba zadat IP adresu serveru, který odkazuje na Apple zařízení s Mac OS.

5. Následně proběhne kompilace aplikace a výsledný IPA balíček bude ve stromové struktuře iOS projektu ve složce Bin.

A.2.5 Vystavení aplikace na App Store

Posledním krokem je samotné vystavení aplikace na App Store, kde následně dojde k ověření a schválení aplikace. K nasazení aplikace bude potřeba program Application Loader, který je spustitelný nově i na Windows a Linux zařízeních. V následujících krocích bude popsáno proces jak použít tento program k samotnému nasazení aplikace.

1. Stáhnout Application Loader na adrese
   `http://www.appuploader.net/appuploader/download.php`
2. Spustit program Application Loader, přihlásit se k účtu a zvolit možnost Deliver Your App.
3. Vybrat cestu k IPA souboru, který reprezentuje implementovanou aplikaci.
4. Potvrzením se odešle aplikace do App Store.
5. Přihlášením do iTunes Connect lze sledovat stav v jakém se aplikace nachází. Doba ověřování aplikace může trvat u více než týden, následně pak bude aplikace přístupná pro veřejnost.



Obrázek 35: Vystavení aplikace pomocí Application Loader.